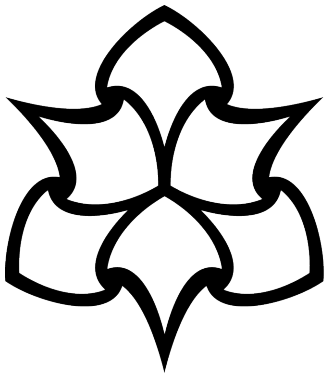


Dynamic Scheduling Algorithm in Cyber Mimic Defense Architecture of Volunteer Computing

QIANMU LI, School of Cyber Science and Engineering, Nanjing University of Science and Technology, Nanjing, Chi
E-mail: qianmu@njust.edu.cn
SHUNMEI MENG, School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China, E-mail: mengshunmei@njust.edu.cn
XIAONAN SANG, 1School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China,2 Intelligent Manufacturing Department, Wuyi University, Jiangmen, China E-mail: 913000720238@njust.edu.cn
HANRUI ZHANG, 1School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China, 2 Jiangsu Zhongtian Internet Technology Co., Ltd. Nantong, 226463, China, E-mail: jessica_9533@njust.edu.cn
SHOUJIN WANG, Department of Computing, Macquarie University, Sydney, Australia, E-mail: shoujin.wang@mq.edu.au
ALI KASHIF BASHIR, Department of Computing and Mathematics, Manchester Metropolitan University, UK, E-mail: Dr.alikashif.b@ieee.org
KEPING YU (CORRESPONDING AUTHOR), Global Information and Telecommunication Institute, Waseda University, Japan, E-mail: keping.yu@aoni.waseda.jp
USMAN TARIQ, College of Computer Science and Engineering, Prince Sattam bin Abdulaziz University, Saudi Arabia, E-mail: u.tariq@psau.edu.sa

XX

Volunteer computing uses computers volunteered by the general public to do distributed scientific computing. *Volunteer computing* is being used in high-energy physics, molecular biology, medicine, astrophysics, climate study, and other areas. These projects have attained unprecedented computing power. However, with the development of information technology, the traditional defense system cannot deal with the unknown security problems of *volunteer computing*. At the same time, Cyber Mimic Defense (CMD) can defend the unknown attack behavior through its three characteristics: dynamic, heterogeneous and redundant. As an important part of the CMD, the dynamic scheduling algorithm realizes the dynamic change of the service centralized executor, which can enusre the security and reliability of CMD of *volunteer computing*. Aiming at the problems of passive scheduling and large scheduling granularity existing in the existing scheduling algorithms, this paper first proposes a scheduling algorithm based on time threshold and task threshold, and realizes the dynamic randomness of mimic defense from two different dimensions; finally, combining time threshold and random threshold, a dynamic scheduling algorithm based on multi-level queue is proposed. The experiment shows that the dynamic scheduling algorithm based on multi-level queue can take both security and reliability into account, has better dynamic heterogeneous redundancy characteristics, and can effectively prevent the transformation rule of heterogeneous executors from being mastered by attackers.

KEYWORDS

Volunteer computing, Cyber Mimic Defense, Dynamic Scheduling, Time Threshold, Task Threshold, Multi-level Queue

1 INTRODUCTION

With the explosive expansion of the global cyberspace, the traditional defense model follows "threat perception, cognitive decision-making, problem removal", which has been unable to resist the known but not occurred security risks and unknown security risks. Therefore, the research goal of network security defense technology should be to find a new way of defense that does not rely on attack characteristics and behavior information, so as to alleviate the growing problem of network space security. In 2014, Wu Jiangxing published the vision of Cyber Mimic Defense (CMD) [1], which is based on the highly available and highly reliable non similar redundancy structure, combined with the multi-mode voting mechanism that does not rely on rules and features to determine, through the nonlinear transformation of several service components with equivalent functions and different structures, to interfere with the behavior decision of attackers. In CMD, in order to make the attacker cannot create the conditions that can invade all the heterogeneous execution components at the same time, a dynamic scheduling strategy that can resist the cooperative attack is introduced, which makes it difficult for the attacker to maintain the attack chain and sniff[2-6].

In recent years, there has been a rapidly-growing interest in *volunteer computing* systems, which allow people from any where on the Internet to contribute their idle computer time towards solving large parallel problems. Probably the most popular examples of these are distributed.net, which gained fame in 1997 by solving the RSA RC5-56 challenge using thousands of volunteers' personal computers around the world, and SETI @ home, which is currently employing hundreds of thousands of volunteer machines to search massive amounts of radio telescope data for signs of extraterrestrial intelligence. A

number of academic projects have also ventured to study and develop *volunteer computing* systems, including some, like our own Bayanihan, that promote web-based systems using Java. Even the commercial sector has joined the fray, with a number of new startup companies seeking to put *volunteer computing* systems to commercial use, and pay volunteers for their computer time. The key advantage of *volunteer computing* over other forms of met a computing is its ease-of-use and accessibility to the general public. By making it easy for anyone – even casual users – on the Internet to join in a parallel computation, *volunteer computing* makes it possible to build very large global computing networks very quickly, as proven by the success of SETI @home and distributed.net.

The most important thing in *volunteer computing* CMD is the dynamic scheduling of heterogeneous executors. It is responsible for organizing and building the current service set of executors, realizing the dynamic change of executor components, and keeping the whole architecture dynamic redundancy. Therefore, the scheduling strategy often determines the security of the whole *volunteer computing* CMD. At present, there are few researches on the dynamic scheduling algorithm, and the proposed scheduling algorithm is either too regular or too random, which cannot take into account the security and reliability of the system [7-10]. Moreover, most algorithms rely on the feedback mechanism of passive scheduling, and the scheduling object granularity is large [11-14]. Therefore, the research of a fine-grained active scheduling algorithm which can not only achieve dynamic randomness, but also ensure the reliability of the system has a guiding role for the further improvement of *volunteer computing* CMD, even for improving the severe security situation in cyberspace.

This paper focuses on the dynamic scheduling algorithm, which is very important in CMD of *volunteer computing*. The purpose of dynamic scheduling is to make the attacker unable to grasp the internal changes of the system through the non-linear transformation of heterogeneous redundancy. At present, most of the existing scheduling strategies are passive scheduling based on the feedback mechanism[15-16], and the scheduling object is often the whole set of executor services, with large scheduling granularity[17-18], and the general scheduling strategies are based on the unified transformation cycle, without strong randomness[19-21], and some scheduling strategies cannot meet the reliability requirements[22]. In addition, there is no research about CMD analyze the research of quantitative model. So combined with the research status of CMD, aiming at the shortcomings of the existing dynamic scheduling algorithm, the main research work of this paper is as follows:

1. Analyzing the quantitative model of security analysis is given for *volunteer computing* CMD architecture through the perspective of intrusion success probability;

2. Aiming at the *volunteer computing* problem that the granularity and passivity of the scheduling algorithm and the randomness of the executor based on the unified transformation cycle, a stochastic threshold based dynamic scheduling algorithm is proposed, which includes time random thresholds and task random thresholds. The main idea is to assign a random threshold for each execution body before work, and when the threshold is reached, the execution body will be transformed. In order to implement this algorithm, ziggurat random threshold generation algorithm is also given, and the concept of random threshold pool is defined to provide threshold resources for the execution body scheduling.

3. In view of the lack of reliability of the scheduling algorithm and the shortcomings of the random threshold scheduling algorithm, a dynamic scheduling algorithm based on multilevel queue is proposed. Firstly, based on the minimum similarity model of the executive body, the initialization algorithm of the execution volume set is given. Secondly, the multi-level feedback queue of the process scheduling is used, combined with the time random threshold and task random threshold. This way takes the security and reliability into account.

This paper first addresses the problem of the lack of reliability of existing scheduling algorithms. Through the minimum similarity model of the executive body, the initialization algorithm of the executive body set is given. Then, in view of the shortcomings of the scheduling algorithm based on random threshold, this paper draws on the multi-level feedback queue of process scheduling, combines the random threshold of time and the random threshold of task, and proposes a dynamic scheduling algorithm based on multi-level queue. Finally, a simulation experiment was done. On the premise of ensuring the low uncertainty of the experiment, this paper has done 150 experiments, and obtained 150 scheduling cycles and 150 common mode failure rates. According to the simulation results of the simulation experiment, this article compares the five mimic defense dynamic scheduling algorithms from two aspects of security and reliability, including the completely random scheduling algorithm (CRS), the normal distribution based scheduling algorithm (NDS) Time-based random threshold scheduling algorithm (TIRTS), task-based random threshold scheduling algorithm (TARTS), multi-level queue scheduling algorithm (MQS). This paper verifies that MQS has higher security and reliability.

In Section 3, this paper introduces the mimic defense architecture; in Section 4, a dynamic scheduling algorithm based on random threshold is proposed, which includes time random threshold and task random threshold; in Section 5, a dynamic scheduling algorithm based on multi-level queue is proposed by combining time random threshold and task random threshold; in Section 6, the advantages and disadvantages of similar algorithms are verified by simulation experiments.

2 RELATED WORKS

This section mainly introduces the research work related to the dynamic scheduling algorithm of Mimic Defense.

As an important part of the mimic defense architecture, the basic function of dynamic scheduling is to organize and build the current set of executor services, and take control of the input agent component to send external input requests to the specified executor, dynamically select the executor components in the service set, and realize the replacement, offline, service migration and other operations of the executor[23]. The main purpose of the dynamic scheduling algorithm is to make the pseudo representation of the execution body in the service set unable to be sniffed, so it is easier to hide the internal characteristics of the target object. Therefore, the advantages and disadvantages of dynamic scheduling algorithm determine

whether the whole architecture has high security and reliability. In recent years, there are more and more research results about dynamic scheduling algorithm in mimic defense.

In reference [24], a completely random scheduling algorithm is proposed, which is also a general method at present. In this strategy, the control parameters are used as random seeds to generate pseudo-random numbers, and the redundancy and number of heterogeneous executors are further determined to schedule. In reference [25], a hard real-time non periodic task fault-tolerant scheduling algorithm is proposed, which reduces the influence of voting, switching and other factors by combining specific voting strategies. It reduces the error rate of scheduling when executing aperiodic tasks. In reference [26], an agent scheduling model based on attack and defense game is proposed. By using the prior probability statistics of attack behavior and combining the differences between heterogeneous agents, the revenue value of both sides of attack and defense and the overall difference value between the service set of the agent and the set to be selected are calculated The optimal scheduling strategy is selected by the Bayesian Stackelberg game model. a random seed minimum similarity algorithm is proposed in reference [27]. Firstly, the construction of heterogeneous executors is analyzed, the similarity between heterogeneous executors is calculated by the similarity between components, and then the overall similarity of the set of executors is obtained. Finally, the scheduling strategy with the minimum overall similarity is selected during scheduling scheme; in reference [28], a scheduling strategy based on normal distribution is proposed. Based on the idea of random scheduling, by introducing the carrier function and defining the attributes of heterogeneous executors, the scheduling strategy can achieve randomness and controllability at the same time.

The above mimic defense scheduling algorithms have their own advantages, but they also have shortcomings. Most of the scheduling algorithms do not fully implement the scheduling process, but how to choose the most appropriate transformation scheme when the executor needs to schedule, when the scheduling is determined by the feedback controller, and the scheduling object is the whole set of heterogeneous executors. So, most of the existing scheduling algorithms are large granular active scheduling. However, the general scheme of active scheduling only makes the set of executors transform periodically based on different time, which is often observed by attackers. While the completely random transformation period cannot guarantee the reliability of the system. Therefore, it is necessary to research a fine-grained scheduling algorithm for a single executor to constrain the scheduling period and ensure the security and reliability.

3 VOLUNTEER COMPUTING CMD

In nature, after a long process of evolution and variation, organisms have gradually evolved their own way of survival. A variety of organisms can use their physiological structure and characteristics to disguise as other creatures in the environment, or integrate with the environment, so as to improve their attack ability or defense ability. This biological phenomenon is called "mimicry phenomenon". Biological mimicry is an inspiration to *volunteer computing* Cyberspace Security Defense, which is called Cyberspace Mimic Defense (CMD). Dynamic Heterogeneous Redundancy (DHR) architecture is the core architecture of CMD system. Therefore, DHR architecture has three core characteristics: dynamic, heterogeneous and redundant.

In DHR architecture, the attack complexity is promoted to the cooperative attack level of "dynamic heterogeneous multiple targets under uncoordinated conditions". This exponential level of attack difficulty makes it difficult for intruders to find a way to escape by means of continuous trial and error elimination. Even if some kind of attack is successful, the dynamic feedback mechanism inside DHR makes it difficult to inherit the previous successful experience. DHR not only improves the uncertainty of attack effect, but also improves the robustness of target system including high reliability, high availability and high credibility. DHR typically constructs a negative feedback control mechanism based on multi-mode voting policy distribution and multi-dimensional dynamic reconstruction [28], as shown in Fig.1.

As can be seen from the above figure, the main function of input agent is policy distribution. According to the instructions of policy scheduling, it determines whether to bind external input with the executor in the current heterogeneous executor set, so as to realize executor activation, pending repair and other tasks. The executor in the current service executor set is represented as A_j (j=1,2,⋯,k), Each service executor A_j can correspond to a variety of functional equivalent to be selected Heterogeneous executors according to the strategy of distribution, which is expressed as E_i (i=1,2,⋯,m); each executor to be selected E_ican realize multi-dimensional dynamic reconstruction technology such as reorganization, reconstruction and redefinition through various software and hardware modules in component pool; The policy scheduler is also a feedback controller. When the feedback information of the multi-mode voter indicates that the output vector is abnormal, the following three steps will as follows:

- Judge whether there is a selectable normal output vector according to the given voting strategy. If not, judge whether other voting algorithms need to be called for secondary voting;
- According to the given scheduling policy instructions, in the policy distribution phase, remove the suspicious executors from the current service set, or replace the executors to be selected into the service set, or directly assign new defense scenarios to the suspicious executors, and even transform the defense scenarios of all executors in the service set if necessary;
- Observe the status between the update of defense scenario and the output voting to determine whether the first two operations need to be repeated.

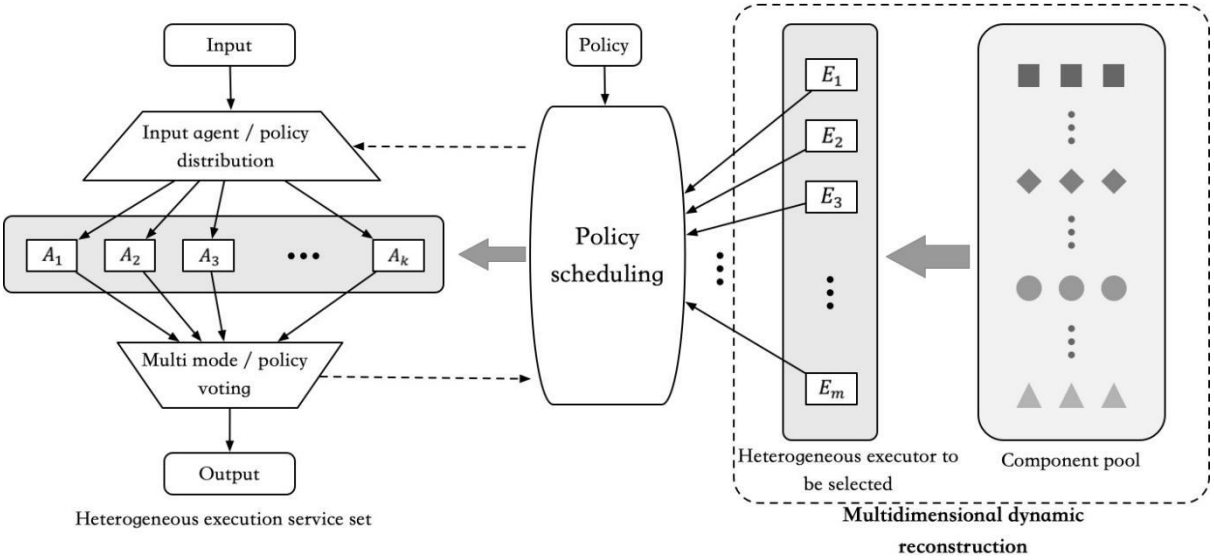


Fig. 1. typical structure of DHR

In the simulation defense system with DHR as the core architecture, the multi-mode voter realizes the uncertainty attribute and fault tolerance ability, and makes up for the defects caused by the certainty and static of the traditional defense system. For the intruder, the mimic defense architecture has the uncertainty effect, because as long as the multi-mode voter finds that the output vector of the executor in a transient stable scenario is abnormal, it will trigger a dynamic negative feedback control mechanism and send out instructions to replace the abnormal executor in the current service set. From the perspective of reliability, the mimic defense architecture has strong fault tolerance. In multi-mode voting, most of the same or identical output vectors are selected from the multi-mode output vectors of the current service set as the output response of the input excitation. Once the dynamic mechanism with closed-loop feedback property is activated, the service structure will be changed or the resources will be reconfigured, so as to eliminate the impact of the abnormal output of the executor in the service set.

In summary, mimic defense has explored a new direction of network security technology research. It not only shows the characteristics of point surface integrated security defense, but also can build a highly reliable service scenario, which is of great significance to change the rules of cyberspace security.

4 DYNAMIC SCHEDULING ALGORITHM BASED ON RANDOM THRESHOLD

4.1 The selection of random threshold

Considering that the probability of random events can be represented by probability distribution, the threshold probability distribution in scheduling strategy can be specified. A series of thresholds obeying a certain probability distribution are selected to indirectly control the transformation rule of heterogeneous executors in the pseudo defense architecture, so that the dynamic and controllability can be considered in the dynamic scheduling strategy based on random thresholds. Considering the continuity of random threshold and the difficulty of selection, this chapter uses probability density function to reflect the distribution of threshold.

Definition 1 random threshold： Select a random number r for an execution T with an attribute A of the Heterogeneous executor as the listening object, When the attribute A of the executive body T reaches the set r, the random number r is called the random threshold of the executive body T based on the attribute A.

Definition 2 random threshold probability density function：The function describing the probability of random threshold near a certain value point is a probability model of selecting random threshold, which reflects the distribution of random threshold. In this paper, two kinds of random thresholds, time and task, are involved, which correspond to the probability density function of time random threshold and the probability density function of task random threshold respectively.

In order to meet the requirements of the system for the randomness and controllability of the dynamic scheduling algorithm, the random threshold probability density function needs to have a broad domain of definition, controllability and complexity. The probability density function of normal distribution meets the above basic characteristics. The long-term practice of industry and academia shows that normal distribution is very similar to the distribution of objective phenomena in many fields. From the perspective of central limit theorem, if a thing is affected by many factors, no matter what the distribution of each factor itself is, the average value of the sum of them is normal distribution [29]：

The probability density function of random threshold is determined as normal function, which means that the selection of random threshold is based on normal distribution, or a series of random thresholds selected in dynamic scheduling algorithm obey normal distribution. According to the probability density function, Ziggurat algorithm [30] is used to select the random threshold.

The specific process of initialization algorithm for heterogeneous executive body sets is as follows:

(1) Calculating a similarity threshold value ϕ_{th};

(2) Randomly select two actuators P_i and P_j $(1 \le i \ne j \le m)$ in the candidate set, if $Record[i][j] > \phi_{th}$, repeat step (2), otherwise, record $P = \{P_i, P_j\}$ $P|_{list} = \emptyset$ and continue;

(3) Select the executors in turn from the set to be selected, if the current executor $P_k (1 \le k \le m)$ meets $P_k \notin P$ and any $P_l \in P$ has $Record[k][l] < \phi_{th}$, then $P = \{P_k\} \cup P$;Repeat step (3) until $|P| = n$;When $|P| = n$, order $P|_{list} = \{P\} \cup P|_{list}$;

(4) Reset $P = \{P_i, P_j\}$ and return to step (3) until a new set of executors $P_{new} (P_{new} \notin P|_{list})$ can no longer be obtained;

(5) The executive body set with the lowest comprehensive similarity in $P|_{list}$ is selected as the initialization executive body set, that is $P_{init} = min\{P \to \Phi | P \in P|_{list}\}$.

In order to facilitate the implementation of the above step (4) and obtain all the execution body sets that are not repeated and meet the conditions, the backtracking method will be adopted in the minimum similarity algorithm. The specific algorithm is described as follows:

ALGORITHM: Heterogeneous Executor Body Set Initialization Algorithm Based on Minimum Similarity

Precondition assumption: similarity between all executants has been calculated and recorded, and a method *calSim* for calculating *n* redundancy executants set has been defined;

Input: heterogeneous executive body set *E* to be selected, total number *m* of all heterogeneous executors, redundancy *n* of pseudo defense architecture, similarity record *Record* between executors;

Output: Heterogeneous Executive Body Set *P*;

Function $MinimumSimilarityForInitialization(E, m, n, Record)$

```
1.  /* Calculate similarity threshold */
2.  threshold ← max(Record) + (max(Record) − min(Record)) * n/m;
3.  /* Randomly select two benchmark actuators */
4.  while True
5.    i ← RandInt(1, m);
6.    j ← RandInt(1, m);
7.    if i == j or Record[i][j] > threshold then
8.      continue;
9.    else
10.     break;
11.   end if
12. end while
13. P ← [E_i, E_j];
14. PList ← [ ];
15. /* Get all the required executive body sets */
16. getPList(2, P, 1);
17. /* Find the executive body set with Least Comprehensive Similarity */
18. minS ← max(Record);
19. minP ← [ ] ;
20. for each PP in PList do
21.   simPP ← calSim(PP)
22.   if simPP < minS then
23.     minS ← simPP;
24.     minP ← PP;
25.   end if
26. end for
27. return minP;
```

// Recursive backtracking to find all the required executive body sets

Function $getPList(num, P, lastIndex)$

```
28. if num == n then
29.   PList.add(P);
30. end if
31. if lastIndex == m then
32.   return;
33. end if
34. /* Select the executors from the set to be selected */
35. for each k in range(lastIndex, m) do
36.   flag ← True;
37.   for each p in P do
```

```
        l ← index(p);
        if Record[k][l] > threshold then
          flag ← False;
          break;
        end if
      end for
      if flag then
        getPList(num + 1, P + [E_k], k);
      end if
    end for
```

Definition 3 random threshold auxiliary function: In the algorithm of random threshold generation, the probability density function of random threshold is f(x), There is a distribution G that is easy to sample, and its probability density function is g(x). If there is a constant $M > 1$, so that there is $f(x) \leq Mg(x)$ in the whole definition domain of x, then g(x) is called random threshold auxiliary function.

The core idea of Ziggurat's random threshold generation algorithm is to change the big into the small, and integrate the effective local into the effective whole. By dividing the random threshold probability density function into a section of curve, the random threshold is selected for each section of curve by the method of refusing sampling, so as to avoid too many invalid calculations. In this paper, the random threshold probability density function is selected as a normal function. Due to the symmetry of the normal function, Ziggurat segmentation is performed with the right half of the normal function as an example, as shown in Fig. 2.

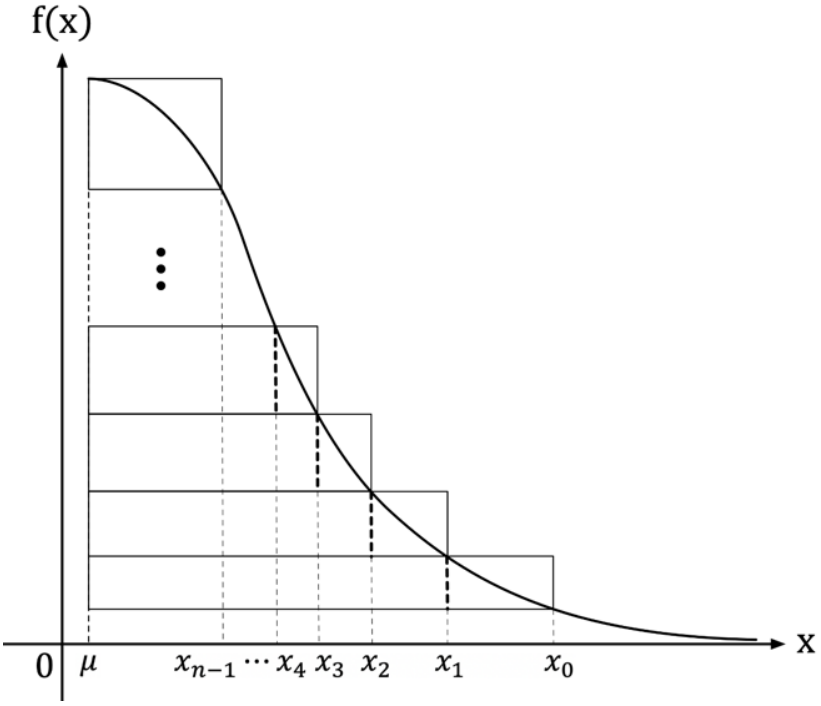


Fig. 2. Ziggurat partition of normal function

In the Ziggurat segmentation graph above, the right half of the normal function is divided into n+1 parts, including the ladder shape composed of n rectangles and the area with the lowest tail, and each partition point (the abscissa of the right vertex of the rectangle) is $x_0, x_1, x_2, \cdots, x_{n-1}$. In addition, the most important point is that the area of these n+1 segmentation part is equal, so as to ensure that the probability of sampling from any one segmentation is equal, and the equal area value is defined by the trailing part.

$$S = (x_0 - \mu) \cdot f(x) + \int_{x_0}^{\infty} f(x)dx \tag{1}$$

However, it is impossible to determine the specific area value through analytical evaluation. In other words, it is impossible to calculate the analytical solution of x_0. Therefore, the numerical solution of x_0 can only be obtained through approximation method: first, assume x_0, calculate the area S according to formula (1), and then calculate $x_1, x_2, \cdots, x_{n-1}$ one by one. If the difference between the rectangular partition area corresponding to x_{n-1} and S exceeds the error value is given, and then the x_0 will be changed until all the partition areas are equal. Once the value of x_0 is determined, all the dividing points can be recorded as data tables. For normal functions, a pair of (n, μ, σ) values correspond to a data table. In the subsequent use, you can directly check the table to get the required dividing points. After Ziggurat segmentation, you can refuse to sample each rectangular segmentation and collect the samples that meet the requirements.

In addition, considering that the range of threshold is the whole real field, the threshold cannot be too small or even less than 0, and the threshold cannot be too large, so it is necessary to limit the selection range of threshold. According to the 3 σ principle of normal distribution [31-32], the probability of random threshold distribution beyond (μ-3σ,μ+3σ) is 0.3%, which is very small, but it is still necessary to exclude these almost impossible thresholds. Of course, when determining the probability distribution function of random threshold, it will make μ-3σ>0. Therefore, the range of 3 σ can not only achieve the completeness of random threshold, but also ensure the rationality of random threshold.

4.2 Dynamic scheduling algorithm based on time random threshold

Time is the most commonly used scheduling threshold in mimic defense architecture, which is called transformation period.

The traditional scheduling strategy is to fix a period value, all the executors change after a period. This method has been gradually abandoned because it is obvious periodicity and easy to be mastered by attackers. In the current common strategies, the transformation period is dynamically adjusted according to the current network environment, but the period is still for all executors. The time random threshold proposed in this section is for each different executor. According to the description of the random threshold in definition 1, this section takes the execution time of the executor as the listening attribute, So the time random threshold is the random threshold based on the execution time

Definition 4 Time Random Threshold: Assign a random threshold t to a Heterogeneous executor T, and replace it when the execution time of the Heterogeneous executor T reaches t. The threshold t is called the random threshold based on the execution time, which is called time random threshold for short.

It is determined that the random threshold is the execution time. Next, it is necessary to set the time random threshold for each executor. In the previous section, the algorithm of generating random threshold has been given, so only the probability density function of time random threshold, i.e. the expectation μ and standard deviation σ of normal function, need to be determined to meet the distribution law of execution time of executors.

Definition 5 A successful attack [33]: In general, an attack means that the attacker's behavior produces an input and output in the system, and a successful attack may involve multiple attacks, resulting in multiple input and output. In the mimic defense architecture, if an attacker exploits a vulnerability in some executors, resulting in consistent abnormal output of these executors, and the abnormal output is voted by the voter, it is considered as a successful attack.

For a single redundancy architecture, the expected transformation period is always shorter than the time taken for an attack to succeed[34-38]. It is as same as multi redundancy architecture. However, the average time to succeed in an attack is longer than that of single redundancy architecture. If an attacker attacks in a cooperative way, or a large number of executors have the same or same origin vulnerabilities, the time to succeed in an attack is the same as that to succeed in a single redundancy architecture. Therefore, the final time threshold is determined in a single redundancy architecture On the whole, the expected model is:

$$\text{Time threshold} = \text{Time to success of an attack} - \text{time when the performer performs a task}$$

Analysis in mathematical sense, First, make the following assumptions and definitions:

- The total number of all Heterogeneous executors is m;
- In the single redundancy architecture, the executive body is $P_i(1 \le i \le m)$, The average time taken for an attack to succeed is $\overline{T}_{attack}^{(i)}(1 \le i \le m)$, executive body P_i the average time taken to perform a task is $\overline{T}_{once}^{(i)}(1 \le i \le m)$, The expected time threshold for the transformation is $\overline{T}_{th}^{(i)} = \overline{T}_{attack}^{(i)} - \overline{T}_{once}^{(i)}, (1 \le i \le m)$;
- There is an objective safety factor $\beta_i(1 \le i \le m)$ for a single actuator $P_i(1 \le i \le m)$. $\beta_i(1 \le i \le m)$ refers to the quantification of the security of each heterogeneous executor. Before the standardized hardware and software modules are used as the optional executors in the system, the quantitative parameters should be given after the security evaluation. To some extent, the security factor will limit the impact of the uncertainty of the success of an attack;
- Setting the artificial control coefficient $\alpha_i(1 \le i \le m)$ for the executor $P_i(1 \le i \le m)$ is a manual setting to control the effect of the time taken by an executor to succeed in an attack on the expectation of the random threshold probability density function;
- In the calculation of the average time threshold, the influence weight of the executor $P_i(1 \le i \le m)$ is the standardized control coefficient $\omega_i(1 \le i \le m)$, and $\omega_i = \frac{\alpha_i\beta_i}{\sum_{k=1}^{m}\alpha_k\beta_k}$, $\sum_{i=1}^{m}\omega_i = 1$, which is determined by the safety factor number and human control coefficient of the executor.

Based on the above assumptions and definitions, the expected value and standard deviation of time random threshold probability density function are:

$$\mu_{time} = \sum_{i=1}^{m} \omega_i \overline{T}_{th}^{(i)} \tag{2}$$

$$\sigma_{time} = \sqrt{\sum_{i=1}^{m} \omega_i \left(\overline{T}_{th}^{(i)} - \mu_{time}\right)^2} \tag{3}$$

Here, $\omega_i = \frac{\alpha_i \beta_i}{\sum_{k=1}^{m} \alpha_k \beta_k}$, $\overline{T}_{th}^{(i)} = \overline{T}_{attack}^{(i)} - \overline{T}_{once}^{(i)}$.

Therefore, the final time random threshold probability density function is expressed as:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma_{time}} e^{-\frac{(x-\mu_{time})^2}{2\sigma_{time}^2}} \tag{4}$$

As a whole, to determine the expectation and standard deviation of the probability density function of time random threshold, it is necessary to conduct attack simulation in the mimic defense architecture with single redundancy to obtain the corresponding attack success time of each executor, as well as the time of one input and output, which can be adjusted by the safety coefficient and human control coefficient of the executor itself. In addition, because the task threshold is an integer value, the final result needs to be rounded in the random threshold generation algorithm.

According to the typical structure of dynamic heterogeneous redundancy in the mimic defense architecture, combined with the definition and generation of time random threshold, a dynamic scheduling model based on time random threshold is constructed, as shown in Fig.3.

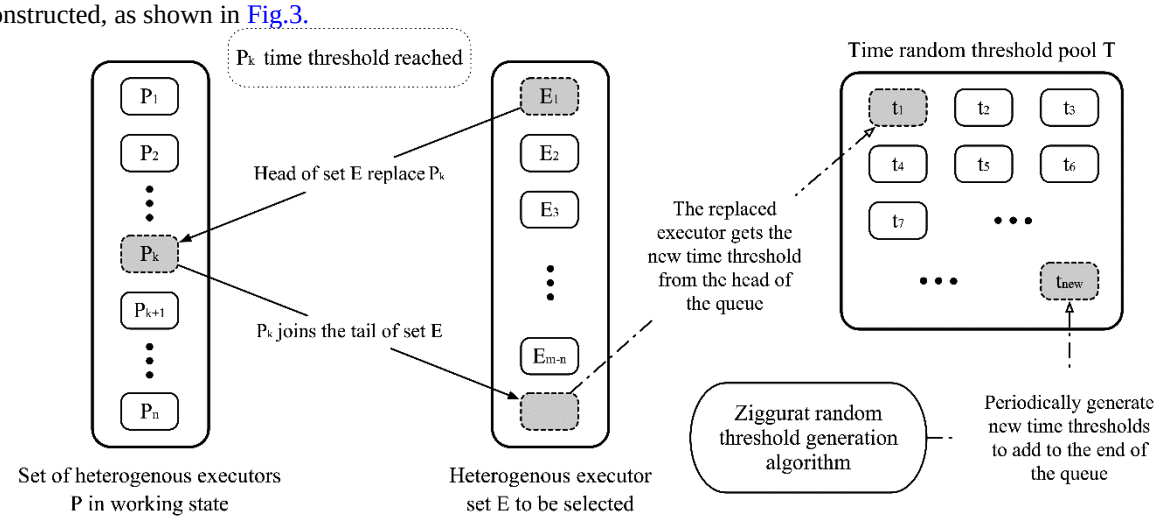


Fig. 3. dynamic scheduling model based on time random threshold

Among them, the total number of all Heterogeneous executors is m, set $P(P_1, P_2, \cdots, P_n)$ is the set of Heterogeneous executors in the working state, set $E(E_1, E_2, \cdots, E_{m-n})$ is the set of Heterogeneous executors to be selected, and set $T(t_1, t_2, \cdots)$ is the time random threshold pool.

Combined with the scheduling model in Fig.3., the dynamic scheduling of heterogeneous executors is illustrated as follows:

- The time random threshold pool is a queue structure independent of the mimic defense architecture. Through Ziggurat random threshold generation algorithm, new elements are added to the tail of the queue periodically. The threshold allocated from the head of the queue no longer exists in the threshold pool;
- The time random threshold pool can ensure that the supply of time threshold is greater than the demand during initialization and operation;
- During initialization, the executors in set P and set E are randomly arranged. In the process of work, set E is a queue structure. When an executor in set P reaches the time threshold, it is added to the end of set E and assigned a new time threshold. At the same time, the first executor in set E is replaced by the vacant position in set P;
- The time random threshold ensures that each replaced Heterogeneous executor can be assigned to a new time threshold at the first time, which avoids the serialization of scheduling and threshold calculation, and improves the overall efficiency of dynamic scheduling algorithm;

The dynamic scheduling algorithm TIRTS based on time random threshold is shown in algorithm 1:

ALGORITHM 1: dynamic scheduling algorithm based on time random threshold (TIRTS)

Assumption： Ziggurat random threshold generation algorithm periodically adds new elements to the time random threshold pool to ensure that the supply of time threshold is always greater than the demand; Set of heterogeneous executors E to be selected，Time random threshold pool T is queue structure; Available listening objects exist;

Input： Sum of Heterogeneous executors m, Redundancy of mimic defense architecture, Set of heterogeneous executors in working state P, Set of heterogeneous executors to be selected E, Time random threshold pool T;

Function : SchedulingBasedOnTimeRandomThreshold(m,n,P,E,T)

/* Initialize the heterogeneous execution set. Currently, P is an empty set, and the number of elements in set E is m */

if*(|P|==0)* **then**

```
    /* Assign random time thresholds to all Heterogeneous executors */
    for each entity in E do
        entity.TimeThreshold ←T.pop();
    end for
    /* Select n executors from the selected set to the service set, and the listener starts */
    for each i in range(n) do
        entity ←E.top();
        entity.pos ←i;
        P.push(entity);
        threshold ←entity.TimeThreshold;
        listener ←entity.TimeListener(threshold,Scheduling);
        listener.start();
end for
end if

/* Function executed after listener triggers (discovery reaches time threshold) */
Function : Scheduling(old_entity):
    /* The executor is added from the service set to the tail of the selection set, and the time threshold is reallocated */
    E.push(old_entity);
    old_entity.TimeThreshold ←T.pop();
    i ←old_entity.pos;
    new_entity ←E.pop();
    new_entity.pos ←i;
    P[i] ←E.pop();
```

4.3 Dynamic scheduling algorithm based on task random threshold

In the previous section, the random threshold is set as the execution time of the executor, and the scheduling based on the time threshold is also based on the traditional transformation cycle. Of course, the attributes that can be used as scheduling threshold in heterogeneous executors are not only execution time. In this section, the number of execution tasks of executors is selected as the listening attribute, and a dynamic scheduling algorithm, TARTs, based on task random threshold is proposed.

Definition 6 Task Random Threshold: A random threshold w is assigned to a heterogeneous executor T. when the number of tasks executed by the executor t reaches w, it is replaced. The threshold w is called the random threshold based on the number of tasks executed by the executor T, which is called task random threshold for short.

The number of tasks to be executed is selected as the random threshold, and then the random threshold for each executor is assigned. Similar to the previous section, based on the random threshold generation algorithm, it is necessary to determine the probability density function of the random threshold of the task, that is, to determine the expectation μ and standard deviation σ of the normal function to meet the distribution law of the number of tasks to be executed by the executor.

In the single redundancy architecture, the number of tasks to be executed is always expected to be less than the number of successful attacks. In the multi redundancy architecture, the average number of successful attacks will be greater than the single redundancy. However, in the case that the attacker adopts the cooperative attack or a large number of executors have the same origin vulnerabilities, the number of successful attacks will be one The number is equivalent to the number in a single redundancy architecture. The above discussion is consistent with that in the previous section. The determination of the final task threshold is also based on the single redundancy architecture. The overall task threshold expectation model is:

$$\text{Task threshold} = \text{number of successful attacks in one attack} - 1$$

Similarly, in a mathematical sense, make the following assumptions and definitions:

- The total number of all Heterogeneous executors is m;

- In the single redundancy architecture, the executor is $P_i(1 \le i \le m)$, the average number of successful attacks is $\overline{W}_{attack}^{(i)}(1 \le i \le m)$, and the expected task threshold of the executor P_i is $\overline{W}_{th}^{(i)} = \overline{W}_{attack}^{(i)} - 1, (1 \le i \le m)$;
- Set the human control coefficient $\alpha_i(1 \le i \le m)$ for the actuator $P_i(1 \le i \le m)$;
- In the calculation of the average task threshold, the influence weight of the executor $P_i(1 \le i \le m)$ is the standardized control coefficient $\omega_i(1 \le i \le m)$, and $\omega_i = \frac{\alpha_i\beta_i}{\sum_{k=1}^{m}\alpha_k\beta_k}$, $\sum_{i=1}^{m}\omega_i = 1$

Based on the above assumptions and definitions, the expected value and standard deviation of the probability density function of task random threshold are:

$$\mu_{task} = \sum_{i=1}^{m} \omega_i \overline{W}_{th}^{(i)} \tag{5}$$

$$\sigma_{task} = \sum_{i=1}^{m} \omega_i \left(\overline{W}_{th}^{(i)} - \mu_{task}\right)^2 \tag{6}$$

And $\omega_i = \frac{\alpha_i\beta_i}{\sum_{k=1}^{m}\alpha_k\beta_k}$, $\overline{W}_{th}^{(i)} = \overline{W}_{attack}^{(i)} - \overline{W}_{once}^{(i)}$

Therefore, the probability density function of the final task random threshold is expressed as:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma_{task}} e^{-\frac{(x-\mu_{task})^2}{2\sigma_{task}^2}} \tag{7}$$

In order to determine the expectation and standard deviation of the probability density function of the random threshold of the task, it is also necessary to simulate the attack in the mimic defense architecture with single redundancy, to obtain the number of successful attacks corresponding to each executor, and to adjust the safety coefficient and human-oriented control coefficient of the executor.

According to the typical structure of dynamic heterogeneous redundancy in the mimic defense architecture, combined with the definition and generation of task random threshold, a dynamic scheduling model based on task random threshold is built, as shown in Fig.4.

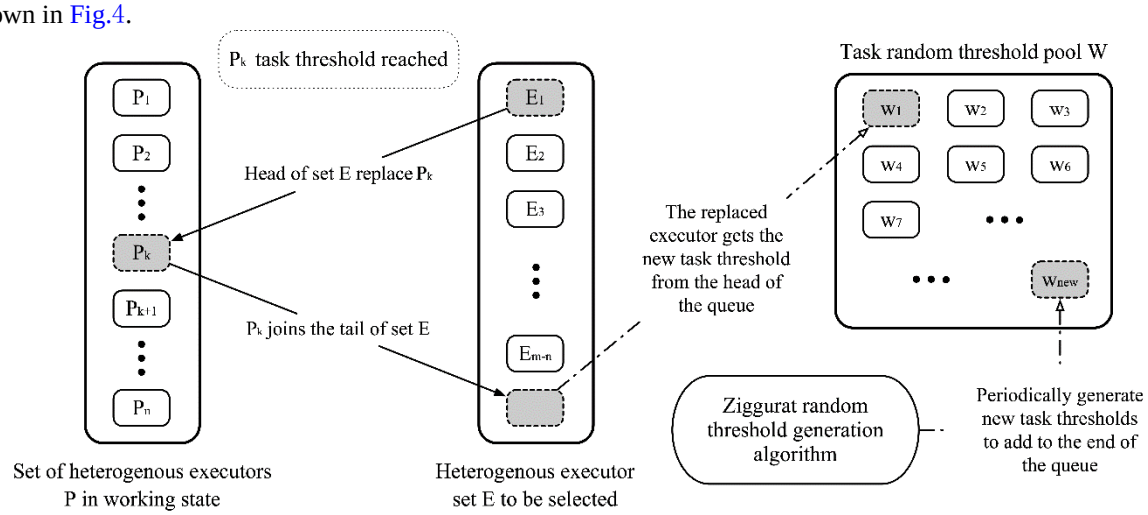


Fig. 4. dynamic scheduling model based on task random threshold

Among them, the total number of all Heterogeneous executors is m, set $P(P_1, P_2, \cdots, P_n)$ is the set of Heterogeneous executors in working state, set $E(E_1, E_2, \cdots, E_{m-n})$ is the set of Heterogeneous executors to be selected, set $W(w_1, w_2, \cdots)$ is the task random threshold pool, which is similar to the time random threshold pool $T(t_1, t_2, \cdots)$. The basic form of the dynamic scheduling algorithm based on task random threshold is the same as that of time random threshold scheduling. Combined with the scheduling model in Fig.4. , the specific process is shown in algorithm 2:

ALGORITHM 2: Dynamic scheduling algorithm based on task random threshold TARTS

Assumption： Ziggurat random threshold generation algorithm periodically adds new elements to the time random threshold pool to ensure that the supply of time threshold is always greater than the demand; Set of heterogeneous executors E to be selected， Time random threshold pool T is queue structure; Available listening objects exist;

Input： Sum of Heterogeneous executors m, Redundancy of mimic defense architecture, Set of heterogeneous executors in working state P, Set of heterogeneous executors to be selected E, Time random threshold pool T;

Function： SchedulingBasedOnTaskRandomThreshold(m,n,P,E,W)

```
/* Initialize the heterogeneous execution set. At this time, P is an empty set, and the number of elements in set E is m */
if(|P|==0)  then
  /* Assign task random thresholds to all heterogeneous performers */
  for each entity in E do
    entity.TaskThreshold ←W.pop();
  end for
  /* Select n executors from the selected set to the service set, and the listener starts */
  for each i in range(n) do
    entity ←E.top();
    entity.pos ←i;
    P.push(entity);
    threshold ←entity.TaskThreshold;
    listener ←entity.TaskListener(threshold, Scheduling);
    listener.start();
  end for
end if

/* Function executed after listener triggers */
Function: Scheduling(old_entity):
  /* The executor is added from the service set to the tail of the selection set, and the task threshold is reassigned */
  E.push(old_entity);
  old_entity.TaskThreshold ←W.pop();
  i ←old_entity.pos;
  new_entity ←E.pop();
  new_entity.pos ←i;
  P[i] ←E.pop();
```

Whether it is time threshold or task threshold, the core idea of scheduling algorithm based on random threshold is consistent, and the implementation process is similar. This paper analyzes the scheduling algorithm based on random threshold from space complexity and time complexity.

1) space complexity

If the redundancy of pseudo defense architecture is n, the space required by heterogeneous executor service sets is $O(n)$.If the total number of all available heterogeneous executants is m, the space required by candidate set of heterogeneous actuators is $O(m)$. The random threshold pool ensures that the threshold can always be provided, and the total number of actuators with a control capacity of 2 times in the implementation process, that is, the required space is $O(2m)$.Overall, the spatial complexity of the dynamic scheduling algorithm based on random threshold is $O(3m+n)$.

2) Time complexity

Ziggurat random threshold generation algorithm is independent of the scheduling based on random threshold, which avoids the need to generate random threshold in the scheduling process through the random threshold pool and greatly reduces the time complexity. The redundancy of the pseudo defense architecture is n, and the total number of all available heterogeneous executants is m, then the time complexity for allocating random thresholds to each heterogeneous executant is $O(m)$, the time complexity for selecting the executant to the service set and starting the listener is $O(n)$, and the transformation of the executant after the listener is triggered is a linear operation with the complexity of $O(1)$.Finally, the time complexity of the dynamic scheduling algorithm based on random threshold is $O(m+n)$.

5 DYNAMIC SCHEDULING ALGORITHM BASED ON MULTILEVEL QUEUE

In order to improve the security of pseudo defense architecture, a dynamic scheduling algorithm based on random threshold is proposed. Although it can realize the dynamic and controllable scheduling of heterogeneous executors, it can meet the security requirements of the architecture to a certain extent. But there are still two shortcomings：

1) Under the action of random threshold, although the transformation of a single heterogeneous executor makes the overall architecture dynamic, if the replaced executor and the original executor have the same origin vulnerabilities during scheduling, the architecture has not changed much for attackers, and the vulnerabilities they have been using still exist, resulting in the transformation of the executor not playing the role of reducing vulnerability attack timeliness at this time, thus the reliability of executor scheduling is not guaranteed.

2) The selection of random threshold is based on normal function, which satisfies the dynamics and has controllability at the same time. Although the threshold range is limited, there is still a situation that some random thresholds are too large, which leads to some heterogeneous executants not being transformed for a long time. In addition, there is a correlation between the time random threshold and the task random threshold, and the use of one threshold alone leads to the inability to adapt to different task loads. The combination of the two can improve the adaptability of the scheduling algorithm, and can also make up for the decrease in randomness caused by the limitation of the threshold range, thus improving the dynamics and reliability of the executants' scheduling.

The first deficiency of the random threshold scheduling algorithm is to maximize the difference between the execution entities during scheduling, in other words, to ensure the heterogeneity of the pseudo defense architecture, so as to improve the reliability. In order to achieve this, it is necessary to quantify the difference between heterogeneous execution bodies, that is, to define the similarity between the execution bodies. The smaller the similarity, the greater the difference between the execution bodies, and vice versa. Therefore, the scheduling algorithm based on multi-level queue proposed in this paper is based on the minimum similarity. In addition to considering the similarity between the two executants in the scheduling process, the similarity index of the set of executants should be minimized during system initialization.

The second deficiency of the scheduling algorithm based on random threshold is although the selection of random threshold limits the scope, there is also a situation that the threshold is too large, resulting in some executants being in a working state for a long time. In addition, because the range of random thresholds is limited, the thresholds are closer to each other as a whole and randomness is reduced. Therefore, this paper proposes a dynamic scheduling algorithm based on multi-level queue according to the inspiration from the multi-level feedback queue scheduling of processors and the minimum similarity description of executants. Compared with random threshold scheduling, it has the following three improvements:

1) The combination of time random threshold and task random threshold not only enables the executor with larger threshold to be transformed in time, but also makes up for the defect of randomness reduction caused by limiting the threshold range.

2) The initialization of heterogeneous executor sets adopts the minimum similarity algorithm, which provides basic reliability guarantee for subsequent scheduling.

3) In the scheduling process, the executor with the lowest similarity to the replaced executor is selected to ensure the reliability of the system.

To sum up, scheduling based on random threshold cannot guarantee the reliability of mimic defense architecture, which is also the defect of most scheduling algorithms. In this chapter, in order to solve the two shortcomings. Firstly, we use the random seed minimum similarity algorithm[5] to define the similarity model for heterogeneous executors, reduce the probability of the same origin vulnerability after a transformation, and then propose a dynamic scheduling algorithm MQS based on multi-level queue, combining the time threshold and task threshold, to reduce the uncertainty of heterogeneous executor scheduling.

5.1 Minimum Similarity and Comprehensive Similarity

Definition 7 Similarity of Execution Entity: In the mimic defense architecture, it represents the degree of difference between heterogeneous executors with equivalent functions. Hereinafter referred to as similarity; The smaller the similarity is, the greater the difference between executors is; otherwise, the smaller the difference is; the similarity can be quantified.

In order to measure the similarity more easily, we need to define the component type and its corresponding instances.

Definition 8 In the mimic defense architecture, the set of component types that make up the functional equivalent executor is $D = \{D_1, D_2, \cdots, D_K\}$, where K is the total number of component types; The instance set corresponding to component type D_j is $C_j = \left\{c_1^{(j)}, c_2^{(j)}, \cdots, c_{L_j}^{(j)}\right\}$, where $1 \le j \le K$, L_j is the number of instances owned by component type D_j ; the set of executors with equivalent functions is $E = C_1 \times C_2 \times \cdots \times C_K$, in which $\times$ is Cartesian product symbol and the total number is $|E| = \prod_{j=1}^{K} L_j$

Based on the above definition, the composition structure of the set of heterogeneous executors is shown in Fig.3.

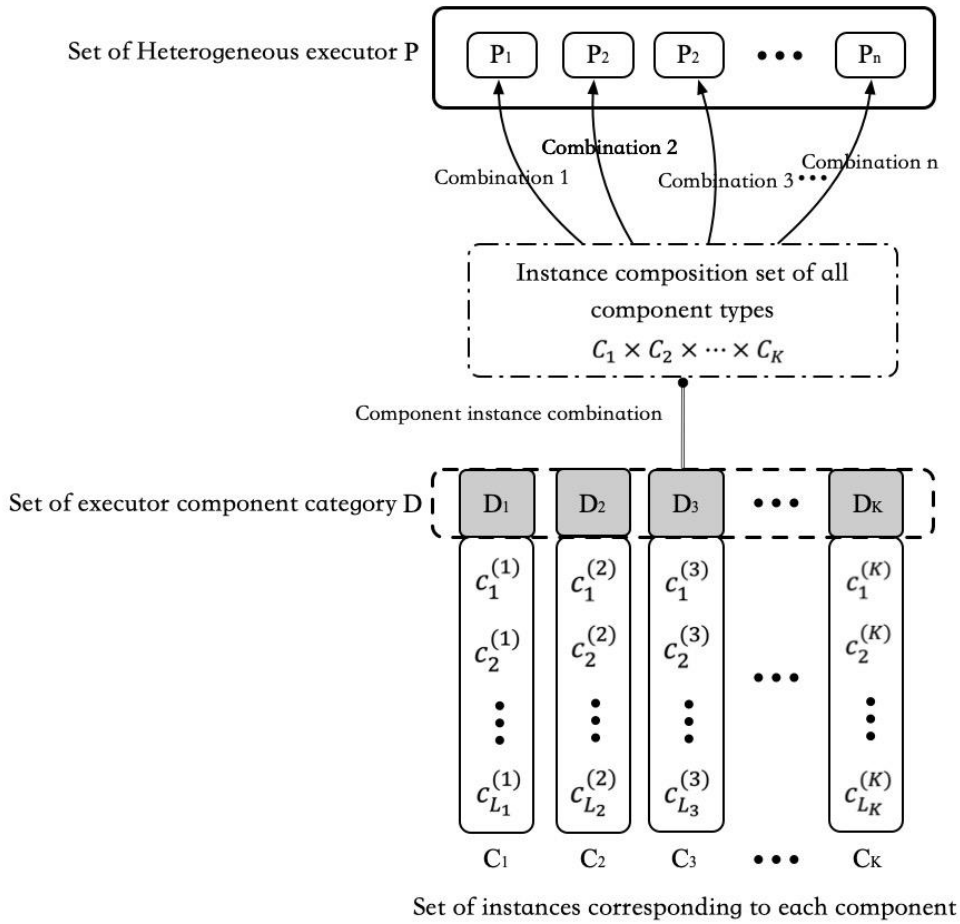


Fig. 5. the structure of the set of heterogeneous executors

For a functional structure, the number of instances of each component will not be large, because it will affect the system power consumption and cost. With the definition of component type and corresponding instance, then we can make quantitative representation for the composition of executor.

Definition 9 In the mimic defense architecture, The total number of Heterogeneous executors is m, Each actuator can be divided into K types of components according to the fixed standard, For the $j(1 \le j \le K)$ component D_j , if the instance selected by the executor $P_i(1 \le i \le m)$ in the corresponding instance set C_j is $c_x^{(j)}$, , it is represented by the eigenvector $Z_j^{(i)} = \left[\theta_1\ \theta_2\ \cdots\ \theta_{L_j}\right]$, and $\theta_l = \begin{cases} 1, & l = x \\ 0, & l \ne x \end{cases}$, $1 \le l \le L_j$; Then, the composition information of the executor P_i can be represented by the feature vector set $Z_i = \left\{Z_1^{(i)}, Z_2^{(i)}, \cdots, Z_K^{(i)}\right\}$

To get the similarity between executors, we need to get the similarity between their component instances.

Definition 10 The vulnerability set of the $p(1 \le p \le L_j)$ of component D_j $(1 \le j \le K)$ is expressed as $V_p^{(j)}$, and the number of containing vulnerability bands is expressed as $n_p^{(j)} = \left|V_p^{(j)}\right|$, and the overall threat level is expressed as:

$$t_p^{(j)} = \sum_{x=1}^{n_p^{(j)}} s_x \tag{8}$$

Two instances of component D_j , $c_p^{(j)}$ $(1 \le p \le L_j)$ and $c_q^{(j)}$ $(1 \le q \le L_j)$ contain a common number of vulnerabilities expressed as $n_{pq}^{(j)} = \left|V_p^{(j)} \cap V_q^{(j)}\right|$, and the threat degree of containing the same vulnerabilities expressed as

$$t_{pq}^{(j)} = \begin{cases} 0, & V_p^{(j)} \cap V_q^{(j)} = \emptyset \\ \sum_{x=1}^{n_{pq}^{(j)}} s_x, & V_p^{(j)} \cap V_q^{(j)} \ne \emptyset \\ 1, & V_p^{(j)} = V_q^{(j)} \end{cases} \tag{9}$$

where s_x is the threat rating of the x-th vulnerability.

The number of component instances in the above definition can be obtained from Common Vulnerabilities and Exposures (CVE); the threat rating of vulnerability can be obtained from Common Vulnerability Scoring System (CVSS), which quantifies the threat rating of the leak to a value between 0 and 10 [32].

Definition 11 The similarity of two examples $c_p^{(j)}(1 \le p \le L_j)$ and $c_q^{(j)}(1 \le q \le L_j)$ of component $D_j(1 \le j \le K)$ is expressed as:

$$\lambda_{pq}^{(j)} = \frac{t_{pq}^{(j)^2}}{t_p^{(j)} \times t_q^{(j)}} \tag{10}$$

From the above definition, we can get that the similarity between two component instances is $0 \le \lambda \le 1$, the smaller the value is, the greater the difference between two component instances is, that is, the greater the degree of heterogeneity; $\lambda = 0$ indicates that two component instances achieve the ideal complete heterogeneity, while $\lambda = 1$ indicates that two component instances are the same in the vast number.

Definition 12 The similarity between instances in component $D_j(1 \le j \le K)$ is expressed as the similarity feature matrix of D_j [5]

$$Y_j = \begin{bmatrix} 1 & \lambda_{12}^{(j)} & \cdots & \lambda_{1L_j}^{(j)} \\ \lambda_{21}^{(j)} & 1 & \cdots & \lambda_{2L_j}^{(j)} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_{Lj1}^{(j)} & \lambda_{Lj2}^{(j)} & \cdots & 1 \end{bmatrix} \tag{11}$$

It can be seen that the similarity feature matrix Y_j of D_j is a symmetric matrix, which contains the similarity between all instances in component D_j, and lays a foundation for calculating the similarity between executors.

Definition 13 The similarity between the heterogeneous executor $P_k(1 \le k \le m)$ and $P_l(1 \le l \le m)$ on the component $D_j(1 \le j \le K)$ is expressed as the product of the corresponding eigenvector and the similarity matrix [5]:

$$\phi_{kl}^{(j)} = Z_j^{(k)} \cdot Y_j \cdot Z_j^{(l)} \tag{12}$$

The similarity between P_k and P_l is expressed as the weighted sum of the similarity between them on each component [5]:

$$\phi_{kl} = \sum_{j=1}^{K} \rho_j \, \phi_{kl}^{(j)} \tag{13}$$

Among them, K is the number of components that make up the heterogeneous executor, ρ_j is the weight of the similarity of component D_j in all components, and $\sum_{j=1}^{K} \rho_j = 1$, which indicates the impact of component D_j on the attack behavior in the composition of executor.

In conclusion, the similarity between two heterogeneous executors is described. Before performing multi-level queue based scheduling, we first calculate the similarity between the two executors and arrange them in order. When scheduling, it is only necessary to select the executor with the least correlation after the transformation instruction is issued.

Since there is no executor in the service set of the executor before the initialization of the mimic defense architecture, the method of scheduling based on random threshold is to select the executor from the waiting set and add it to the service set. But this does not guarantee better heterogeneity of the service set. Therefore, according to the similarity between heterogeneous executors defined in the previous section, this section gives the overall comprehensive similarity of heterogeneous executor service set.

For n redundancy architecture, there are n functional equivalent heterogeneous executors and n (n-1) / 2 similarity between the two, so the comprehensive similarity of heterogeneous executor service set is defined as follows.

Definition 14 The comprehensive similarity of service set of executor in n redundancy architecture is expressed as the mean value of similarity among all executors in the set [31]

$$\Phi = \frac{2}{n(n-1)} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \phi_{ij} \tag{14}$$

Where ϕ_{ij} is the similarity between the executors P_i and P_j

Generally, the executor combination scheme with the lowest overall similarity will be considered in the initialization of the executor service set, but this may not be the optimal scheme. In addition to the smaller overall similarity, the similarity between local executors should also be considered. In order to better illustrate this situation, an example is given next.

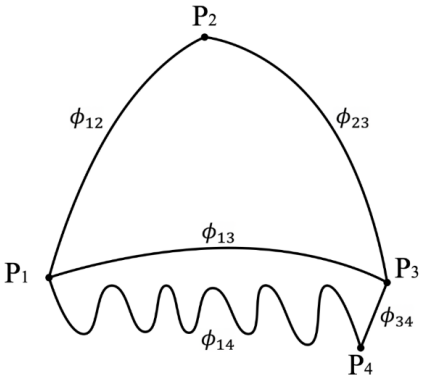


Fig. 6 Actuator and similarity diagram

As shown in Fig. 6, in the 3-redundancy architecture, candidate set includes four actuators P_1、P_2、P_3、P_4, the connection length between the actuators represents the size of the difference, corresponding to ϕ_{12}、ϕ_{23}、ϕ_{13}、ϕ_{14}、ϕ_{34} five similarities respectively. The longer the connection, the smaller the similarity, and $\phi_{12}+\phi_{23}+\phi_{13}>\phi_{13}+\phi_{14}+\phi_{34}$. The two initialized executive body combination schemes are shown in Fig. 7 respectively, and each scheme is composed of a bold execution body point and a similarity dotted line. The execution body combination in scheme a is $\{P_1, P_2, P_3\}$, and the execution body combination in scheme b is $\{P_1, P_3, P_4\}$. Because of $\phi_{12}+\phi_{23}+\phi_{13}>\phi_{13}+\phi_{14}+\phi_{34}$, the execution body set similarity in scheme b is smaller than that in scheme a. However, at this time, it does not mean that scheme b is adopted for initialization, because it can be seen from the figure that the corresponding connection line ϕ_{34} is very short, which means that the similarity ϕ_{34} between the executor P_3 and P_4 is large, so the P_3 and P_4 does not have sufficient difference, and there are loopholes of the same origin with high probability, which are easier to be exploited by attackers.

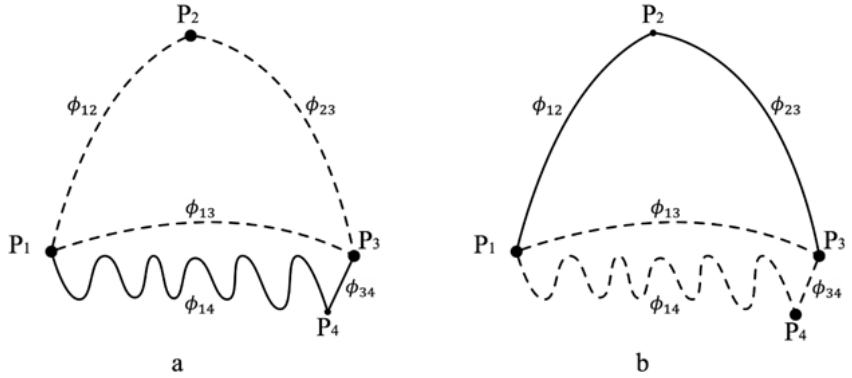


Fig. 7 Combination scheme of two actuators

To sum up, the description definition of similarity of heterogeneous executor sets does not mean that only the executor combination scheme with the lowest overall similarity needs to be directly selected during initialization, and the situation that the similarity between local executors is too small also needs to be avoided. Therefore, it is necessary to set a similarity threshold before initialization to exclude the occurrence of two actuators with smaller similarity. Of course, it is very inefficient to combine all the schemes from the candidate set and then check whether they meet the setting in turn. According to the random seed minimum similarity algorithm, only one executor needs to be randomly selected as the initial benchmark.

5.2 Quasi-Static Measurements: MOKE and MFM

According to the previous description and definition, the following three points should be guaranteed during the initialization of heterogeneous executor set:

- Select benchmark executor and similarity threshold;
- The similarity between two selected executors is less than the set threshold.
- The overall similarity of the service set of the executor is the minimum

In order to compare the similarity and threshold between executors more conveniently, firstly, calculate the similarity between two executors according to formula (13), record and save it, adopt sequence storage structure and record it asRecord, and $\mathrm{Record}[i][j]=\phi_{ij}$, and ϕ_{ij} is the similarity between P_i and P_j expressed by formula (13). $1 \le i \ne j \le m$, m is the total number of all optional executors; meanwhile, the maximum similarity is ϕ_{max} , and the minimum similarity is ϕ_{min}

The random seed minimum similarity algorithm sets a random seed executor, which reduces the calculation amount to a certain extent through preliminary screening. In order to further reduce the initialization work without increasing the

complexity of the algorithm, this section selects two random Executors as the benchmark, and the similarity threshold is set as:

$$\phi_{th} = \phi_{min} + \frac{n}{m} \cdot (\phi_{max} - \phi_{min}) \tag{15}$$

the value of ϕ_{th} will increase with the increase of n, and $\phi_{min} < \phi_{th} \leq \phi_{max}$. While retaining the threshold restriction, it can also guarantee the existence of the initializable set of executors.

The specific process of initialization algorithm of heterogeneous executor set is as follows:

1) Calculate similarity threshold ϕ_{th}.
2) Randomly select two executors P_i and P_j $(1 \leq i \neq j \leq m)$, if $Record[i][j] < \phi_{th}$, repeat step 2, otherwise record $P = \{P_i, P_j\}$, $P|_{list} = \emptyset$ and continue.
3) Select the executor from the set to be selected, If the current executor $P_k (1 \leq k \leq m)$ satisfies $P_k \notin P$, and any $P_l \in P$ has $Record[k][l] < \phi_{th}$, then $P = \{P_k\} \cup P$; repeat step 3 until $|P| = n$; when $|P| = n$, make $P|_{list} = \{P\} \cup P|_{list}$.
4) Reset $P = \{P_i, P_j\}$ and turn back to step 3, until you can no longer get a new set of executors $P_{new} (P_{new} \notin P|_{list})$
5) The set of executors with the least comprehensive similarity in $P|_{list}$ is selected as the initial executor set, that is, $P_{init} = \min\{P \to \Phi | P \in P|_{list}\}$

5.3 Multilevel Queue Scheduling of Executor

In order to adapt dynamic scheduling to different network load, we can use multi-level queue to combine time threshold and task threshold to improve the adaptability and randomness of scheduling algorithm.

According to the inspiration from the multi-level feedback queue of process scheduling, the following definitions are given:

Definition 15 Heterogeneous execution multi-level queue: It is composed of several queues of fixed size and equal in sequence, which are recorded as MQ, and each level of queue can be regarded as a heterogeneous execution service set; hereinafter referred to as multi-level queue;

Definition 16 Multi level queue redundancy: The total number of queues included in the whole multi-level queue, that is, the number of levels of the multi-level queue, is recorded as qn;

Definition 17 Queue capacity: The total number of executors that can be accommodated in each layer of the multi-level queue is recorded as qc, and the capacity of all queues is equal to the redundancy of the mimic defense architecture($qc = n$);

Definition 18 Each level of the multi-level queue is recorded as $Q_1, Q_2, \cdots, Q_{qn}$, and exists $Q_{qn+1} = Q_1$; The heterogeneous executors in each level of queue are represented by queue level sequence number $sn (1 \leq sn \leq qn)$ and position k in the queue, which is recorded as $P_k^{(sn)}$;

The rules for the construction of multi-level queues scheduled by heterogeneous executors are as follows:

Rule 1: Each level of queue is assigned a random threshold of time, and the threshold is small to large with the level from top to bottom;

Rule 2: Time random threshold of all executors sharing queue in the same level queue;

Rule 3: At the same time, only one queue can be used as the set of heterogeneous executors;

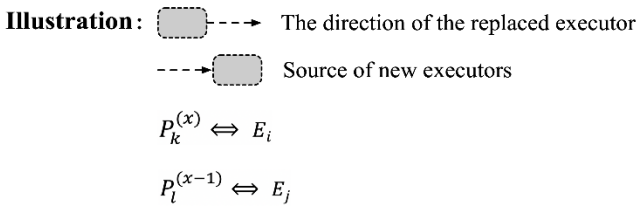
Rule 4: The first level queue is used to initialize the heterogeneous execution set;

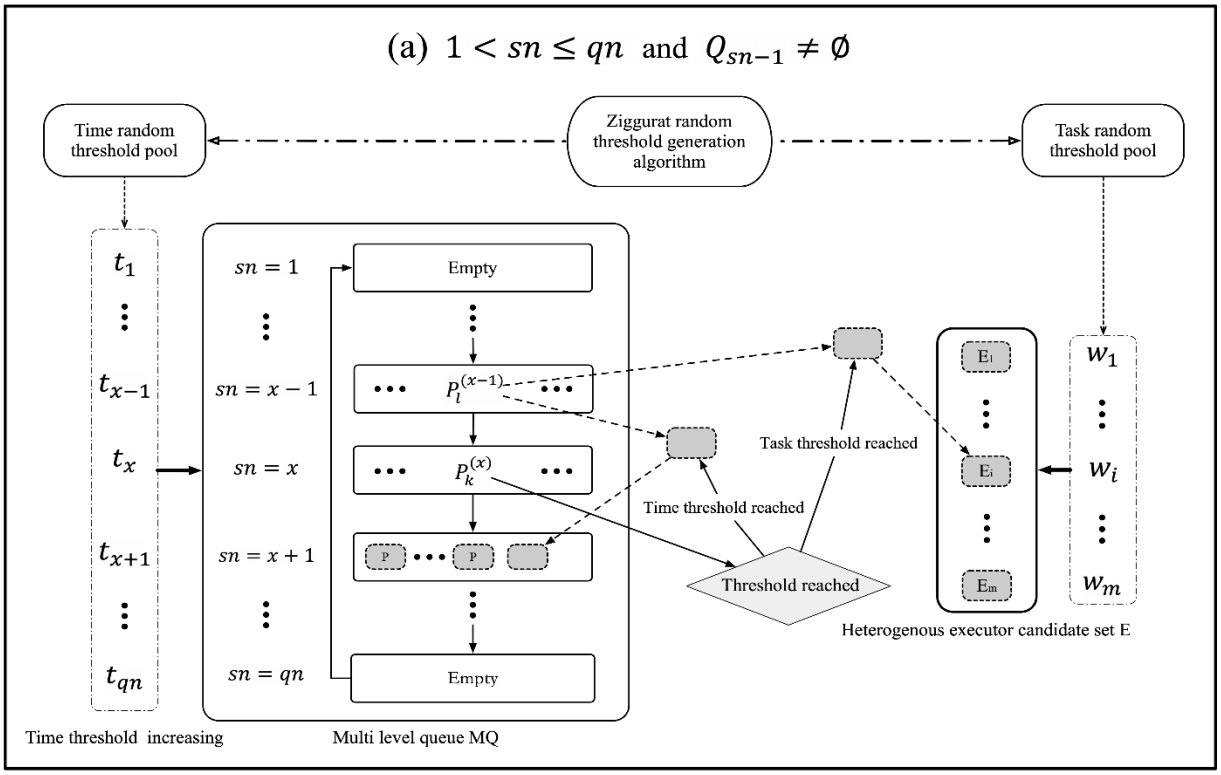
Rule 5: When the next level queue reaches the queue capacity, the heterogeneous execution set becomes;

Rule 6: As long as the upper level queue is not empty, it will be the executor candidate set of the current heterogeneous executor set;

Rule 7: As long as the next level queue does not reach the capacity of the queue, it will be regarded as the storage set of the replaced executor in the current heterogeneous executor set;

Rule 8: The next level queue of the last level queue is the first level queue, while the first level queue does not have the previous level queue.





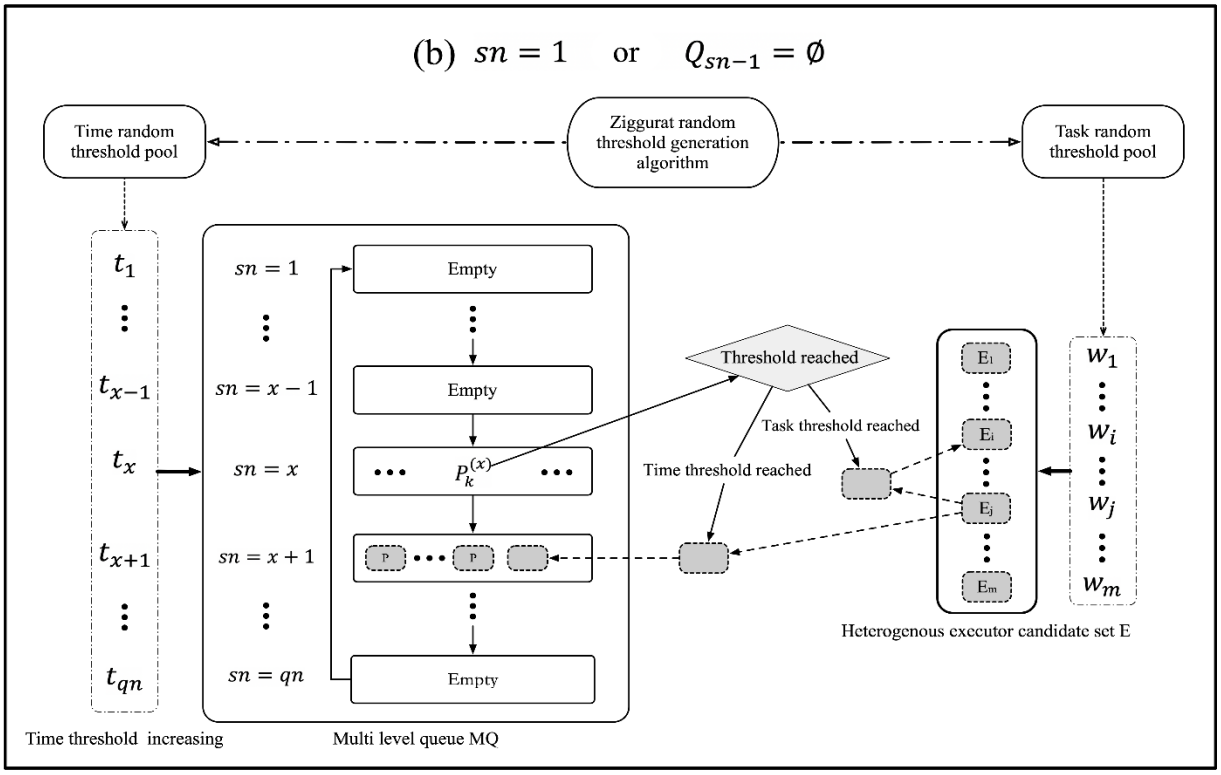


Fig. 8. dynamic scheduling model based on multi-level queue

The dynamic scheduling model based on multi-level queue is shown in Fig.8., assuming that the similarity between two executors has been calculated and recorded as Record, The executors in the selected set E are clearly numbered and the location is fixed during scheduling. The scheduling algorithm flow is as follows:

Select qn time thresholds from the time random threshold pool and sort them in ascending order of size to get $t_1, t_2, \cdots, t_{qn}$, and then allocate them to each level of queue in MQ , $Q_1, Q_2, \cdots, Q_{qn}$.

From the task random threshold pool, select m task thresholds $w_1, w_2, \cdots, w_{qn}$, and assign them to all executors $E_1, E_2, \cdots, E_m$ in the executor to be selected set E.

Through the initialization algorithm of heterogeneous execution set based on the minimum similarity, the first level queue Q_1 is initialized to heterogeneous execution service set, and the current work queue level sn is set to 1.

When $P_k^{(sn)}$ does not reach the time threshold on Q_{sn}, but reaches its task threshold:

if $sn = 1$ or $Q_{sn-1} = \emptyset$, Select the executor $E_j (1 \le j \le m)$ from the set E to replace $P_k^{(sn)}$;

if $1 < sn \le qn$ and $Q_{sn-1} \ne \emptyset$, Select the executor $P_l^{(sn-1)} (1 \le l \le n)$ from Q_{sn-1} to replace $P_k^{(sn)}$.

Make Record[i][j] the smallest available similarity, and $P_k^{(sn)}$ return to the ith position in E and reassign the task threshold

When $P_k^{(sn)}$ does not reach its task threshold w, but first reaches the time threshold on Q_{sn}, the number of tasks completed by $P_k^{(sn)}$ is v($v < w$):

if $sn = 1$ or $Q_{sn-1} = \emptyset$, Select the executor $E_j (1 \le j \le m)$ from the set E to replace $P_k^{(sn)}$;

if $1 < sn \le qn$ and $Q_{sn-1} \ne \emptyset$, Select the executor $P_l^{(sn-1)} (1 \le l \le n)$ from Q_{sn-1} to replace $P_k^{(sn)}$.

Make Record[i][j] the smallest of the available similarity, and add $P_k^{(sn)}$ to the end of Q_{sn+1}, and its task threshold in Q_{sn+1} is $w - v$.

When $|Q_{sn+1}| = qc$ (queue capacity is reached), if $sn < qn$, the task threshold of all executors in Q_{sn} is adjusted to the difference between the initial task threshold and the number of tasks they have completed, and the hierarchical sequence number sn of the work queue is set to $sn + 1$; if $sn = qn$, all executors in Q_{sn} return to the corresponding positions in the selected set E and reassign the task threshold, and sn is set to 1; if $sn = 1$, perform step 1; return to step 4.

This paper analyzes the dynamic scheduling algorithm based on the multi-level queue in terms of space complexity and time complexity:

1) space complexity

If the redundancy of pseudo defense architecture is n, the redundancy of the multi-level queue is qn,the space required by the multi-level queue is $O(n \times qn)$.If the total number of all available heterogeneous executants is m, the space required by candidate set of heterogeneous actuators is $O(m)$. The random threshold pool comprises a time threshold pool and a task threshold pool, and each threshold pool needs twice the capacity of the total number of executants, i.e. the total required space is $O(4m)$. In the algorithm of initialization of executing body set, the record of similarity between heterogeneous actuators requires $O(m^2)$ space. All executable sets that meet the requirements have at most C_m^n and the space complexity is $O(n \times 2^m)$. Overall, the space complexity of the dynamic scheduling algorithm based on multi-level queue is $O(5m + n \times qn + m^2 + n \times 2^m)$.

2) Time complexity

Similarly, Ziggurat random threshold generation algorithm is independent of scheduling based on multi-level queues, and the time complexity will not affect the scheduling algorithm. The redundancy of the pseudo defense architecture is n, the redundancy of the multi-level queues is qn, and the total number of all available heterogeneous executants is m, then the time complexity of assigning a task random threshold to each heterogeneous executant is $O(m)$, the time complexity of assigning a time random threshold to each layer of queues is $O(qn)$, and the time complexity of starting a listener for an executor in the service set is $O(n)$. In the initialization algorithm of executing body set, the time complexity of finding all executing body sets satisfying the requirements by backtracking method is $O(2^m)$. In the scheduling process, the transformation of the executor after the listener triggers needs to find the executor with the smallest similarity with the current executor from the temporary candidate set according to the similarity record, and the time complexity is $O(m)$.Finally, the time complexity of the dynamic scheduling algorithm based on multi-level queue is $O(m + n + qn + 2^m)$.

6 EXPERIMENT ANALYSIS

This experiment includes security verification and reliability verification, which are respectively reflected by the scheduling cycle and the common mode failure rate of the architecture. In order to reduce the uncertainty of the experiment, 150 scheduling cycles and common mode failure efficiency are obtained through 150 tests, and one test is a scheduling cycle.

In the experiment, the redundancy of the mimic defense architecture is 4, 5 and 6 respectively, which can not only compare five scheduling algorithms under different redundancy, but also reflect the relationship between the mimic defense redundancy and security, reliability. In the experiment, there are 12 heterogeneous executors to be selected, which are identified as $E_1, E_2, \cdots, E_{12}$. The specific steps of the experiment are as follows:

6.1 Determine probability density function random threshold

The random threshold pool is required in the TIRTS, TARTS, MQS algorithms. Therefore, two random threshold probability density functions of time and task are determined first. The related prior values are shown in Table 1.

Table 1. RELATED PRIOR VALUES OF HETEROGENEOUS EXECUTOR

	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}	E_{11}	E_{12}
$\overline{T}_{th}^{(i)}$	53.34	43.72	31.55	13.58	32.92	47.53	29.53	44.11	9.71	25.38	57.2	58.61
$\overline{W}_{th}^{(i)}$	20	14	24	13	4	3	4	29	15	23	19	23
α_i	0.156	0.459	0.407	0.204	0.103	0.239	0.432	0.357	0.261	0.149	0.475	0.278
β_i	0.880	0.658	0.986	0.745	0.638	0.566	0.694	0.709	0.859	0.512	0.739	0.797
ω_i	0.052	0.115	0.153	0.058	0.025	0.052	0.114	0.097	0.086	0.029	0.134	0.085

Combining the prior values in the above table, according to formula (2) and (3), two parameters of the probability density function of time random threshold are calculated as $\mu_{\text{time}} = 38.57$ and $\sigma_{\text{time}} = 15.26$. Therefore, the probability density function of time random threshold is

$$f(x)_{time} = \frac{1}{\sqrt{2\pi} \times 15.26} e^{-\frac{(x-38.57)^2}{2\times 15.26^2}} \tag{16}$$

According to formulas (5) and (6), the two parameters of probability density function of task random threshold are $\mu_{\text{task}} = 17.06$ and $\sigma_{\text{task}} = 7.91$, Therefore, the probability density function of random threshold is:

$$f(x)_{task} = \frac{1}{\sqrt{2\pi} \times 7.91} e^{-\frac{(x-17.06)^2}{2\times 7.91^2}} \tag{17}$$

With the normal function expression of two thresholds, two random threshold pools are continuously expanded for threshold selection through Ziggurat random threshold generation algorithm in subsequent experiments.

6.2 Determine heterogeneous executor similarity

In MQS algorithm, the similarity between different executors is needed for the initialization of heterogeneous executor service set and the selection of new executors when scheduling.

In reference [12], it is pointed out that β - distribution adapts to multiple distributions by selecting reasonable parameters and easily obtains calculation results. Therefore, using β - distribution with parameters (5,15) to randomly generate similarity between 12 executors, the similarity matrix is recorded as Record.

In the subsequent simulation scheduling, Record is used to calculate the comprehensive similarity and select the new executor with the least similarity.

$$\begin{pmatrix}
(1, & 0.204, & 0.263, & 0.139, & 0.272, & 0.308, & 0.103, & 0.431, & 0.094, & 0.276, & 0.267, & 0.193) \\
(0.204, & 1, & 0.235, & 0.256, & 0.483, & 0.230, & 0.152, & 0.511, & 0.115, & 0.231, & 0.282, & 0.289) \\
(0.263, & 0.235, & 1, & 0.294, & 0.394, & 0.362, & 0.270, & 0.302, & 0.204, & 0.422, & 0.191, & 0.173) \\
(0.139, & 0.256, & 0.294, & 1, & 0.361, & 0.334, & 0.398, & 0.490, & 0.219, & 0.330, & 0.280, & 0.460) \\
(0.272, & 0.483, & 0.394, & 0.361, & 1, & 0.172, & 0.366, & 0.289, & 0.170, & 0.210, & 0.266, & 0.114) \\
(0.308, & 0.230, & 0.362, & 0.334, & 0.172, & 1, & 0.186, & 0.262, & 0.226, & 0.264, & 0.057, & 0.203) \\
(0.103, & 0.152, & 0.270, & 0.398, & 0.366, & 0.186, & 1, & 0.295, & 0.333, & 0.319, & 0.295, & 0.268) \\
(0.431, & 0.511, & 0302, & 0.490, & 0.289, & 0.262, & 0.295, & 1, & 0.262, & 0.223, & 0.333, & 0.379) \\
(0.094, & 0.115, & 0.204, & 0.219, & 0.170, & 0.226, & 0.333, & 0.262, & 1, & 0.195, & 0.235, & 0.437) \\
(0.276, & 0.231, & 0.422, & 0.330, & 0.210, & 0.264, & 0.319, & 0.223, & 0.195, & 1, & 0.083, & 0.415) \\
(0.267, & 0.282, & 0.191, & 0.280, & 0.266, & 0.057, & 0.295, & 0.333, & 0.235, & 0.083, & 1, & 0.197) \\
(0.193, & 0.289, & 0.173, & 0.460, & 0.114, & 0.203, & 0.268, & 0.379, & 0.437, & 0.415, & 0.197, & 1)
\end{pmatrix}$$

6.3 Scheduling cycle with different redundancy

In the security verification, the scheduling cycle of each scheduling algorithm is obtained through multiple simulation scheduling. Then through the security quantification model of the mimic defense architecture in Chapter 3, the average scheduling cycle is used to calculate the theoretical security metrics of each scheduling algorithm. The so-called scheduling cycle does not refer to the time, but the number of scheduling times used by the heterogeneous execution service set when it becomes the initial set again, regardless of the location of the execution body, For example, the initial service set of the executor is (E_1, E_2, E_3), after x times of scheduling, the service set becomes (E_2, E_3, E_1). Then one cycle of scheduling is $x - 1$. In addition, the five scheduling algorithms in this experiment are all fine-grained scheduling algorithms for a single execution body, so one-time scheduling is one-time execution body scheduling

Redundancy n=4. The scheduling cycle results of the five scheduling algorithms are shown in Fig.9.
Redundancy n=5. The scheduling cycle results of the five scheduling algorithms are shown in Fig.10.
Redundancy n=6. The scheduling cycle results of the five scheduling algorithms are shown in Fig.11.

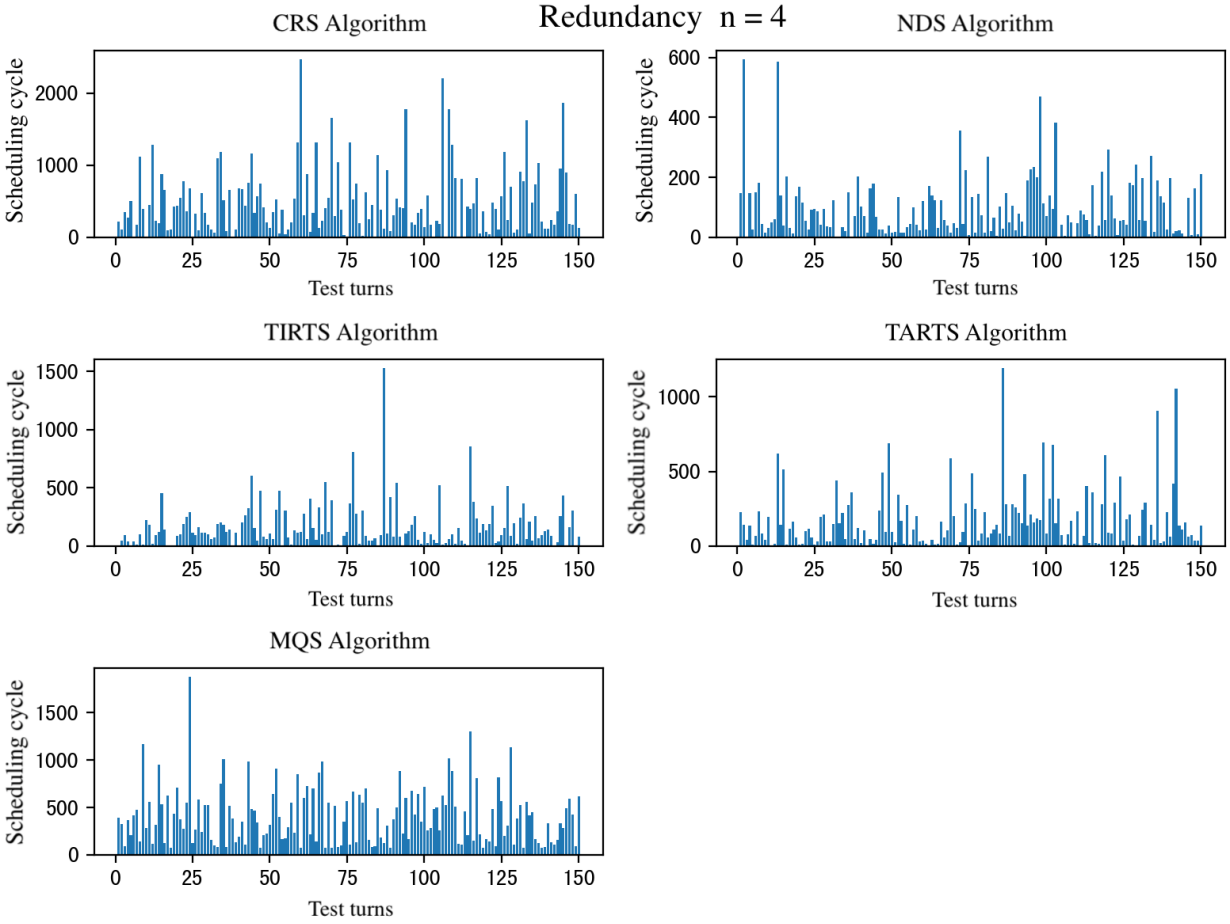


Fig. 9. simulation results of five scheduling algorithms with redundancy n=4

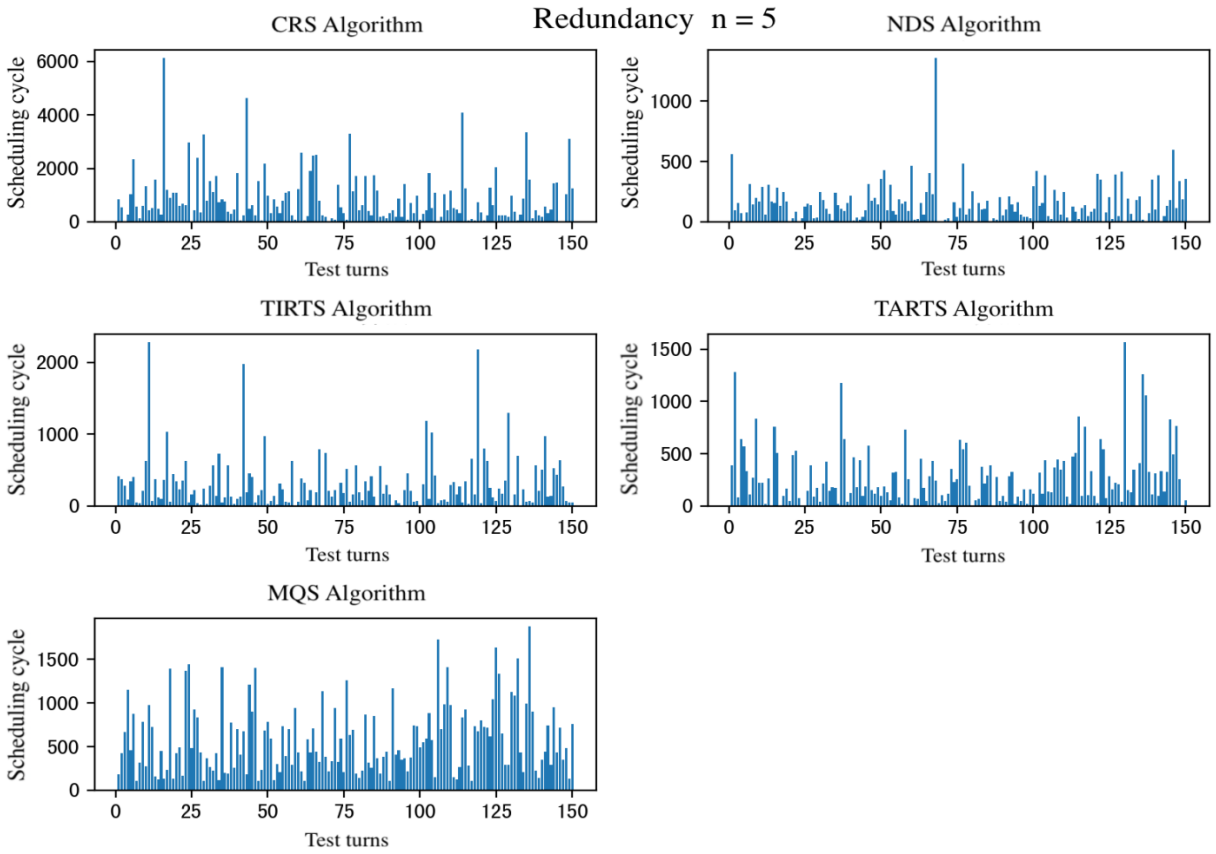


Fig. 10. simulation results of five scheduling algorithms with redundancy n=5

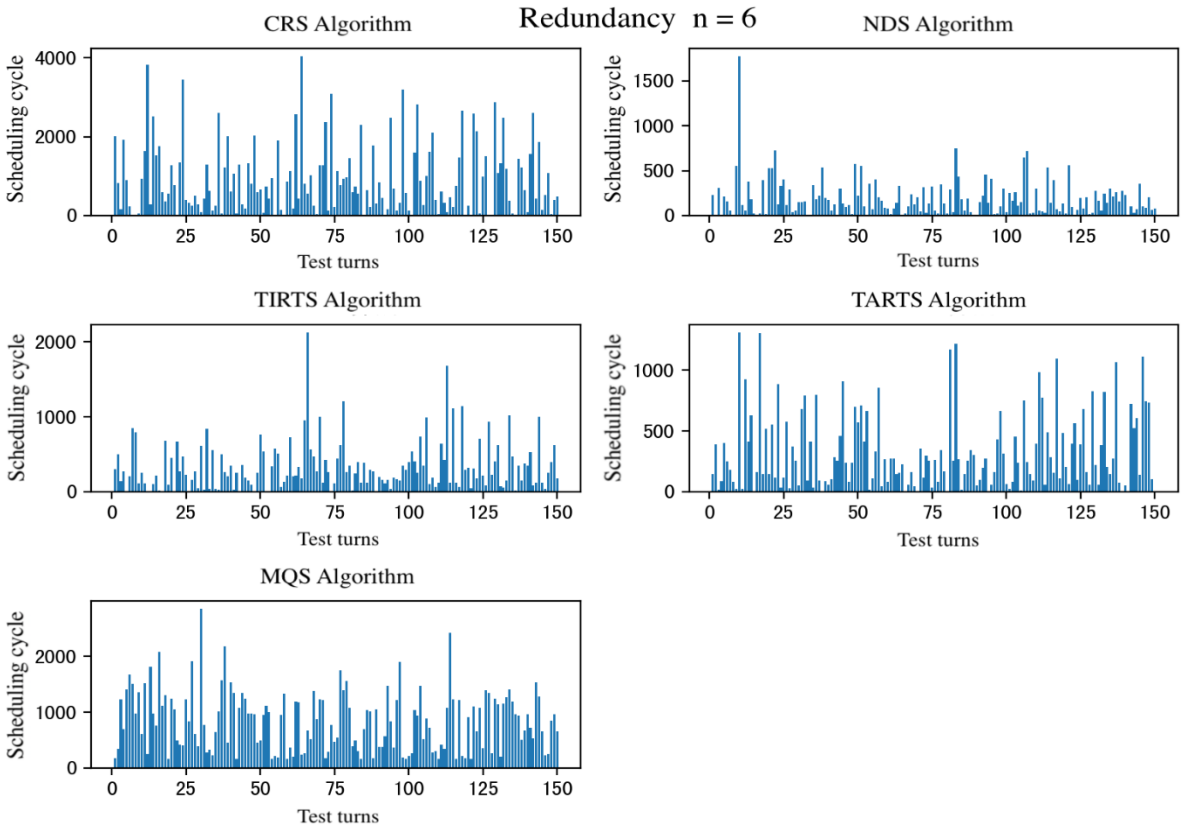


Fig. 11. simulation results of five scheduling algorithms with redundancy n=6

Table 2. Average scheduling period scheduling algorithms with different redundancy

Redundancy \ Algorithm	CRS	NDS	TIRTS	TARTS	MQS
n=4	488.52	95.43	162.99	171.71	404.55
n=5	873.93	154.50	303.33	278.88	578.10
n=6	925.01	191.49	324.12	322.56	843.22

Under different redundancy, the average scheduling period of the five scheduling algorithms is shown in Table 2.

6.4 Architecture inefficiency in scheduling process with different redundancy

In the reliability verification, the common mode failure efficiency of the mimic defense architecture reflects the reliability. The common mode failure efficiency Λ is related to the failure rate λ of a single executor and the similarity between executors ϕ_{ij}, and the specific relationship is as follows [13]:

$$\Lambda = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \left[\left(\lambda_i \cdot \lambda_j \cdot \phi_{ij}\right) / \sum_{k \neq i,j} \sum_{l=k+1}^{n} \phi_{kl} \right] \tag{18}$$

During the experiment, the similarity between the executors can be obtained through the similarity matrix Record in step 2; the failure rate of each executor is randomly generated by the uniformly distributed U(0.01,0.1), and the failure rate sequence of the executors is $F = \begin{pmatrix} 0.0824,0.0088,0.0574,0.0052,0.0747,0.0353, \\ 0.0756,0.0257,0.0425,0.0780,0.0652,0.0632 \end{pmatrix}$. In addition, there will be multiple failure rates in each scheduling cycle, so the common mode failure efficiency of the architecture corresponding to a scheduling cycle is the average of all the common mode failure efficiency in the cycle.

Redundancy n=4. Under the five scheduling algorithms, the failure rate of the architecture is shown in Fig.12.

Redundancy n=5. Under the five scheduling algorithms, the failure rate of the architecture is shown in Fig.13.

Redundancy n=6. Under the five scheduling algorithms, the failure rate of the architecture is shown in Fig.14.

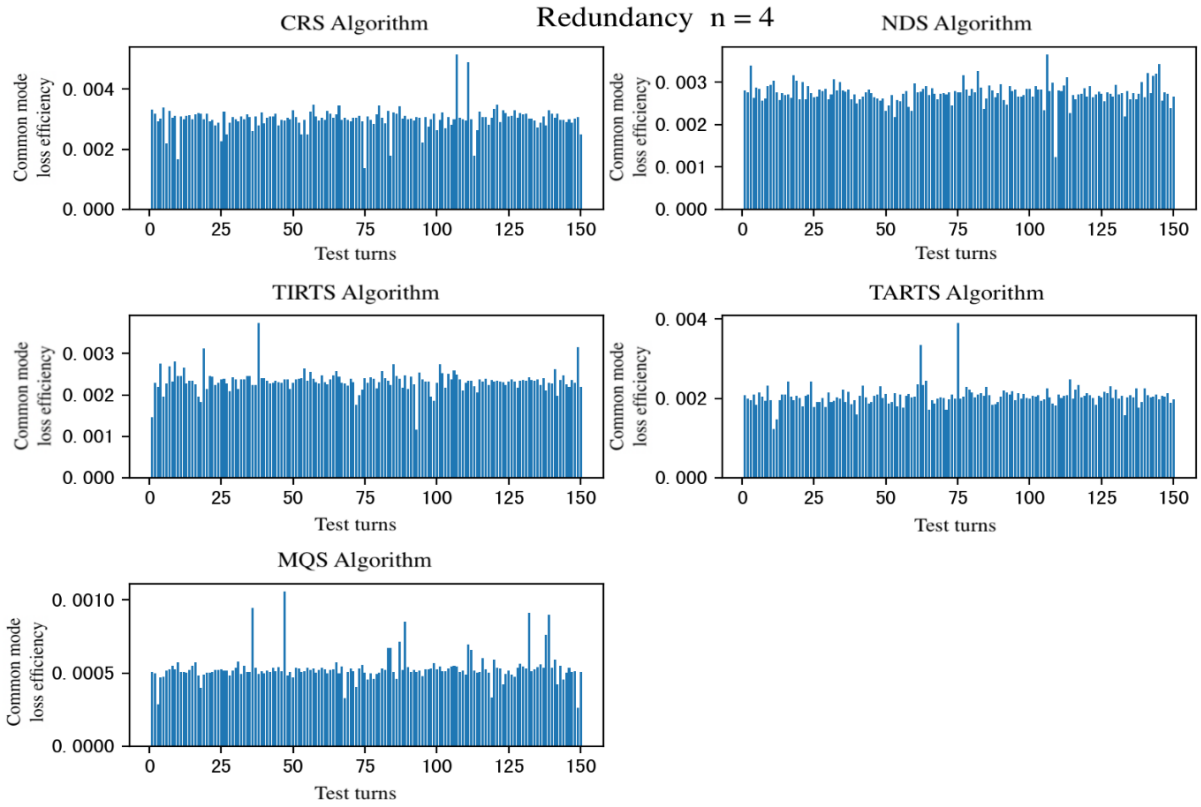


Fig. 12. simulation results of five scheduling algorithms with redundancy n =4

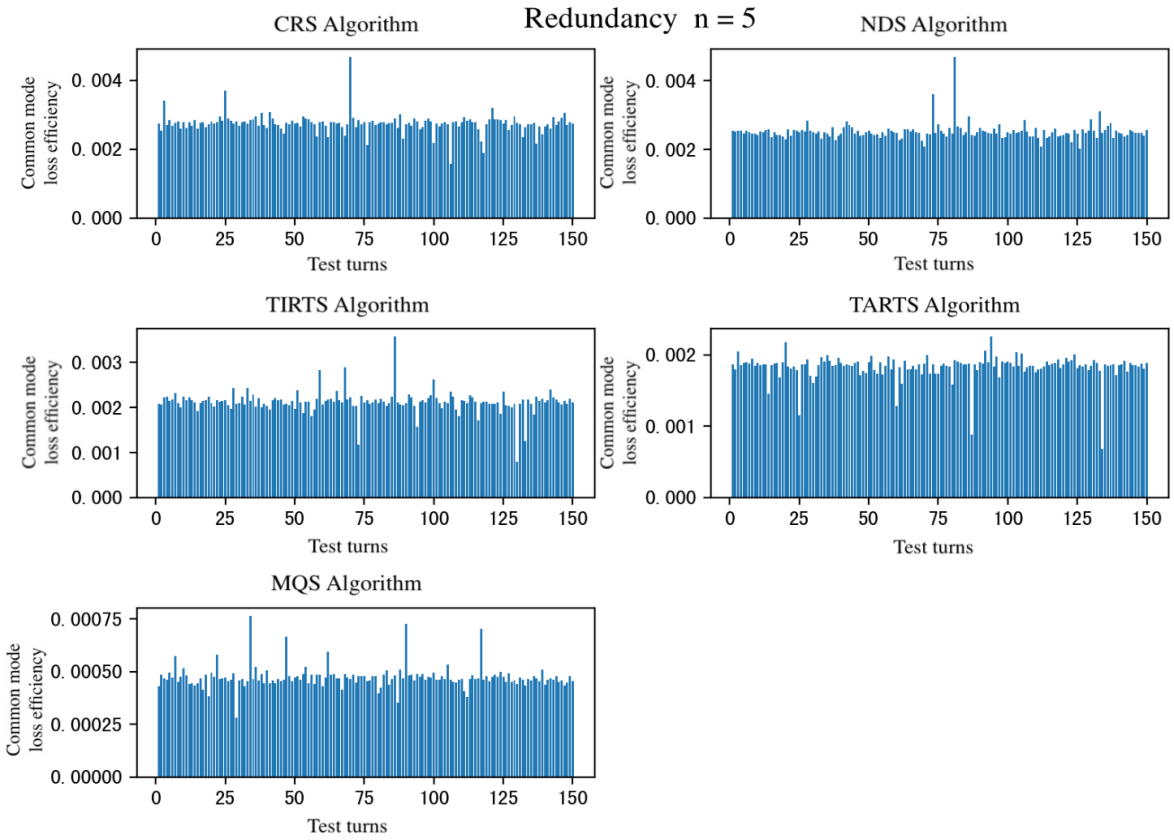


Fig. 13. simulation results of five scheduling algorithms with redundancy n =5

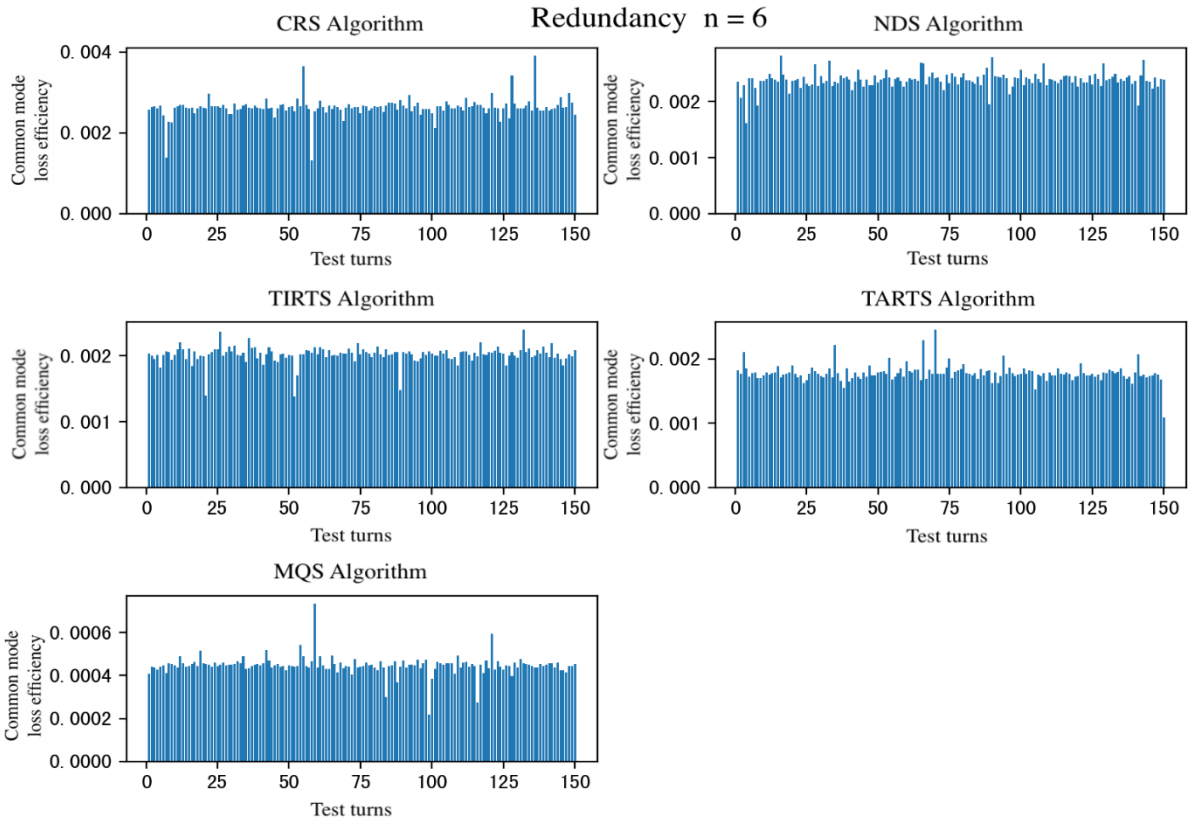


Fig. 14. simulation results of five scheduling algorithms with redundancy n = 6

Table 3. Average common mode loss efficiency of five sechduling algorithms with different redundancy

Redundancy \ Algorithm	CRS	NDS	TIRTS	TARTS	MQS
n=4	0.301%	0.274%	0.234%	0.205%	0.053%
n=5	0.275%	0.252%	0.213%	0.183%	0.047%
n=6	0.263%	0.239%	0.202%	0.178%	0.045%

Under different redundancy, the average common mode failure efficiency of the corresponding architectures of the five scheduling algorithms is shown in Table 3.

From the perspective of safety, Fig.9. to Fig.11. and table II can get the following conclusions:

In terms of security, the TIRTS algorithm, TARTS algorithm and MQS algorithm proposed in this paper are significantly better than NDS algorithm, but less than CRS algorithm.

No matter how much redundancy is, the security of CRS algorithm is the best, followed by MQS algorithm, but with the increase of redundancy, the security of MQS algorithm is close to CRS algorithm.

Under the same scheduling algorithm, the higher the redundancy is, the higher the security is. From the perspective of reliability, the common mode failure efficiency reflects the probability of common mode failure of the mimic defense architecture. The lower the common mode failure efficiency is, the higher the reliability of the algorithm is Fig.12. to Fig.14. and Table 3 can get the following conclusions:

CRS algorithm, NDS algorithm, TIRTS algorithm and TARTS algorithm have higher common mode failure rate and lower reliability.
CRS algorithm has the lowest reliability, MQS algorithm has the highest reliability, and has a significant improvement compared with other algorithms.
Under the same scheduling algorithm, the higher the redundancy is, the higher the reliability is.

7 CONCLUSION

In this paper, a dynamic scheduling algorithm based on random threshold is proposed for *volunteer computing* mimic defense architecture, including TIRTS algorithm and TARTS algorithm, which can ensure certain security and carry out fine-grained active scheduling at the same time. Furthermore, this paper proposes a scheduling algorithm MQS based on multi-level queue, which combines time threshold and task threshold through multi-level queue to improve the security and reliability.

In view of the current passive situation in CMD architecture of *volunteer computing* that the scheduling algorithm only relies on feedback control instructions and the granularity of scheduling is large, we allocate the time threshold or task threshold to each executive body to achieve the active scheduling of a single executive. In order to further improve the adaptability and stability of the random threshold scheduling, and get inspiration from the multi-level feedback queue of the processor, this paper combines the time threshold and task threshold into the multi-level queue, while retaining the advantages of the TIRTS algorithm and the TARTS algorithm, as well as the security and reliability.

8 ACKNOWLEDGEMENTS

This work was supported in part by The 4th project "Research on the Key Technology of Endogenous Security Switches" (2020YFB1804604) of the National Key R&D Program "New Network Equipment Based on Independent Programmable Chips" (2020YFB1804600), the 2020 Industrial Internet Innovation and Development Project from Ministry of Industry and Information Technology of China, Jiangsu province key research and development programs: social development project (BE2017739) , 2018 Jiangsu Province Major Technical Research Project "Information Security Simulation System", the Fundamental Research Fund for the Central Universities (30918012204, 30918014108), National Natural Science Foundation of China (61702264, 61761136003), Postdoctoral Science Foundation of China (2019M651835), and the Japan Society for the Promotion of Science (JSPS) Grants-in-Aid for Scientific Research (KAKENHI) under Grant JP18K18044.

REFERENCES

[1] Wu J. Meaning and vision of mimic computing and mimic security defense. Telecommunications Science, 2014, 30(7): 2-7.
[2] Qi L, Zhang X, Dou W, et al. A distributed locality-sensitive hashing-based approach for cloud service recommendation from multi-source data. IEEE Journal on Selected Areas in Communications, 2017, 35(11): 2616-2624.
[3] Liu H, Kou H, Yan C, et al. Link prediction in paper citation network to construct paper correlation graph. EURASIP Journal on Wireless Communications and Networking, 2019, 2019(1): 1-12.
[4] Al-Turjman F. Intelligence and security in big 5G-oriented IoNT: An overview. Future Generation Computer Systems, 102, 357-368, 2020.
[5] Li Q, Meng S, Wang S, et al. CAD: Command-level anomaly detection for vehicle-road collaborative charging network. IEEE Access, 2019, 7: 34910-34924.
[6] Gong W, Qi L, Xu Y. Privacy-aware multidimensional mobile service quality prediction and recommendation in distributed fog environment. Wireless Communications and Mobile Computing, Article ID 3075849, 8 pages, 2018.
[7] Chi X, Yan C, Wang H, et al. Amplified LSH-based Recommender Systems with Privacy Protection. Concurrency and Computation: Practice and Experience, 2020. DOI: 10.1002/CPE.5681.
[8] Li Q, Meng S, Zhang S, et al. Complex attack linkage decision-making in edge computing networks. IEEE Access, 2019, 7: 12058-12072.
[9] Al-Turjman F, Zahmatkesh H, Al-Oqily I, et al. Optimized Unmanned Aerial Vehicles Deployment for Static and Mobile Targets' Monitoring. Computer Communications, 2020, 149: 27-35.
[10] Xu X, He C, Xu Z, et al. Joint optimization of offloading utility and privacy for edge computing enabled IoT. IEEE Internet of Things Journal, 2019. DOI: 10.1109/JIOT.2019.2944007.

[11] Li Q, Tian Y, Wu Q, et al. A Cloud-Fog-Edge Closed-Loop Feedback Security Risk Prediction Method. IEEE Access, 2020, 8: 29004-29020.
[12] Xu X, Liu Q, Luo Y, et al. A computation offloading method over big data for IoT-enabled cloud-edge computing. Future Generation Computer Systems, 2019, 95: 522-533.
[13] Li Q, Wang Y, Pu Z, et al. Time series association state analysis method for attacks on the smart internet of electric vehicle charging network. Transportation Research Record, 2019, 2673(4): 217-228.
[14] Xu X, Li Y, Huang T, et al. An energy-aware computation offloading method for smart edge computing in wireless metropolitan area networks. Journal of Network and Computer Applications, 2019, 133: 75-85.
[15] Wan S, Zhao Y, Wang T, et al. Multi-dimensional data indexing and range query processing via Voronoi diagram for internet of things. Future Generation Computer Systems, 2019, 91: 382-391.
[16] Li Q, Meng S, Zhang S, et al. Safety risk monitoring of cyber-physical power systems based on ensemble learning algorithm. IEEE Access, 2019, 7: 24788-24805.
[17] Xu X, Xue Y, Qi L, et al. An edge computing-enabled computation offloading method with privacy preservation for internet of connected vehicles. Future Generation Computer Systems, 2019, 96: 89-100.
[18] Gao Z, Xuan H Z, Zhang H, et al. Adaptive fusion and category-level dictionary learning model for multiview human action recognition. IEEE Internet of Things Journal, 2019, 6(6): 9280-9293.
[19] Qi L, Dou W, Wang W, et al. Dynamic mobile crowdsourcing selection for electricity load forecasting. IEEE Access, 2018, 6: 46926-46937.
[20] Wan S, Gu Z, Ni Q. Cognitive computing and wireless communications on the edge for healthcare service robots. Computer Communications, 2020, 149: 99-106.
[21] S Ding, S Qu, Y Xi, S Wan. A long video caption generation algorithm for big video data retrieval. Future Generation Computer Systems, 93, 583-595, 2019.
[22] Wan S, Goudos S. Faster R-CNN for multi-class fruit detection using a robotic vision system. Computer Networks, 2020, 168: 107036.
[23] Al-Turjman F, Nawaz M H, Ulusar U D. Intelligence in the Internet of Medical Things era: A systematic review of current and future trends. Computer Communications, 2019.
[24] Xue C, Lin C, Hu J. Scalability analysis of request scheduling in cloud computing. Tsinghua Science and Technology, 2019, 24(3): 249-261.
[25] Shen D, Luo J, Dong F, et al. VirtCo: joint coflow scheduling and virtual machine placement in cloud data centers. Tsinghua Science and Technology, 2019, 24(5): 630-644.
[26] Liu L, Chen X, Lu Z, et al. Mobile-edge computing framework with data compression for wireless network in energy internet. Tsinghua Science and Technology, 2019, 24(3): 271-280.
[27] Qi L, Dou W, Zhou Y, et al. A context-aware service evaluation approach over big data for cloud applications. IEEE Transactions on Cloud Computing, 2015.
[28] WU J. Research on Cyber Mimic Defense. Journal of Cyber Security, 2016, 1(04):1-10.
[29] Ramasubramanian S. A characterisation of the normal distribution. Annals of the Institute of Statistical Mathematics, 1985, 47(3):410-414.
[30] Marsaglia G, Tsang W W. The ziggurat method for generating random variables. Journal of statistical software, 2000, 5(8): 1-7.
[31] Younis A, Malaiya Y K, Ray I. Evaluating CVSS Base Score Using Vulnerability Rewards Programs. Proc of IFIP International Information Security and Privacy Protection. Springer International Publishing, 2016: 62-75. doi:https://doi.org/10.1007/978-3-319-33630-5_5.
[32] Wang S, Cao L. Inferring implicit rules by learning explicit and hidden item dependency. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2020, 50(3): 935-946.
[33] Wu K, Zhou X Z, Wang J Y, et al. A concept semantic similarity algorithm based on bayesian estimation. Journal of Chinese Information Processing, 2010, 24(2):52-57.
[34] Zhang S, Xiao F, Xu J, Li J. Determination of Aviation Spare Parts Failure Rate Based on Similarity System Theory and Bayesian Theory. Electronics Optics & Control,2015,22(04):83-87
[35] Hou J, Li Q, Cui S, et al. Low-cohesion differential privacy protection for industrial internet. The Journal of Supercomputing, 2020: 1-23.
[36] Li Q, Hou J, Meng S, et al. GLIDE: A Game Theory and Data-Driven Mimicking Linkage Intrusion Detection for Edge Computing Networks. Complexity, vol. 2020, Article ID 7136160, 18 pages, 2020. https://doi.org/10.1155/2020/7136160.
[37] Qianmu Li, Yaozong Liu, Shunmei Meng, Hanrui Zhang, Haiyuan Shen and Huaqiu Long. A dynamic taint tracking optimized fuzz testing method based on multi-modal sensor data fusion. EURASIP Journal on Wireless Communications and Networking (2020) 2020:110. https://doi.org/10.1186/s13638-020-01734-0.
[38] Wang S, Hu L, Wang Y, et al. Modeling Multi-Purpose Sessions for Next-Item Recommendations via Mixture-Channel Purpose Routing Networks. Proceedings of the 28th International Joint Conference on Artificial Intelligence, 2019: 3771-3777.